Certyfikowany Tester

Poziom Podstawowy

(CTFL)

Sylabus

wersja 4.0.1

wersja tłumaczenia PL 4.0.1.0

International Software Testing Qualifications Board

ISTQB
International Software
Testing Qualifications Board

PQB
POLISH
QUALITY BOARD

Informacja o prawach autorskich

© International Software Testing Qualifications Board (zwana dalej „ISTQB®”).

ISTQB® jest zarejestrowanym znakiem towarowym International Software Testing Qualifications Board.

Prawa autorskie © 2024 autorzy sylabusa poziomu podstawowego v4.0.1: Renzo Cerquozzi, Wim Decoutere, Jean-François Riverin, Arnika Hryszko, Martin Klonk, Meile Posthuma, Eric Riou du Cosquer (przewodniczący), Adam Roman, Lucjan Stapp, Stephanie Ulrich (wiceprzewodnicząca), Eshraka Zakaria.

Prawa autorskie © 2023 autorzy sylabusa w wersji 4.0: Renzo Cerquozzi, Wim Decoutere, Klaudia Dussa-Zieger, Jean-François Riverin, Arnika Hryszko, Martin Klonk, Michaël Pilaeten, Meile Posthuma, Stuart Reid, Eric Riou du Cosquer (przewodniczący), Adam Roman, Lucjan Stapp, Stephanie Ulrich (wiceprzewodnicząca), Eshraka Zakaria.

Prawa autorskie © 2019 autorzy aktualizacji 2019: Klaus Olsen (przewodniczący), Meile Posthuma i Stephanie Ulrich.

Prawa autorskie © 2018 autorzy aktualizacji 2018: Klaus Olsen (przewodniczący), Tauhida Parveen (wiceprzewodnicząca), Rex Black (kierownik projektu), Debra Friedenberg, Matthias Hamburg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh i Eshraka Zakaria.

Prawa autorskie © 2011 autorzy aktualizacji 2011: Thomas Müller (przewodniczący), Debra Friedenberg i ISTQB WG Foundation Level.

Prawa autorskie © 2010 autorzy aktualizacji 2010: Thomas Müller (przewodniczący), Armin Beer, Martin Klonk i Rahul Verma.

Prawa autorskie © 2007 autorzy aktualizacji 2007: Thomas Müller (przewodniczący), Dorothy Graham, Debra Friedenberg i Erik van Veenendaal.

Prawa autorskie © 2005 autorzy Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson i Erik van Veenendaal.

Prawa autorskie wersji polskiej: Copyright © Polish Quality Board (PQB).

Tłumaczenie z języka angielskiego: Adam Roman.
Przegląd tłumaczenia: Adam Ścierski.
Przegląd tłumaczenia i korekta: Monika Petri-Starego.

Wszelkie prawa zastrzeżone. Autorzy niniejszym przenoszą prawa autorskie na ISTQB®. Autorzy (jako obecni właściciele praw autorskich) i ISTQB® (jako przyszły właściciel praw autorskich) uzgodnili następujące warunki użytkowania:

- Fragmenty niniejszego dokumentu mogą być kopiowane do użytku niekomercyjnego, pod warunkiem podania źródła. Każdy akredytowany dostawca szkoleń może wykorzystać niniejszy sylabus jako podstawę kursu szkoleniowego, pod warunkiem podania autorów i ISTQB® jako źródła i właścicieli praw autorskich do sylabusa oraz pod warunkiem, że wszelkie reklamy takiego kursu szkoleniowego mogą zawierać wzmiankę o sylabusie dopiero po otrzymaniu oficjalnej akredytacji materiałów szkoleniowych od uznanej przez ISTQB® rady krajowej.
- Każda osoba lub grupa osób może wykorzystać niniejszy sylabus jako podstawę artykułów i książek, pod warunkiem podania autorów i ISTQB® jako źródła i właścicieli praw autorskich do sylabusa.
- Wszelkie inne wykorzystanie niniejszego sylabusa jest zabronione bez uprzedniej pisemnej zgody ISTQB®.
- Każda rada krajowa uznana przez ISTQB® może przetłumaczyć niniejszy sylabus, pod warunkiem, że w przetłumaczonej wersji sylabusa zamieści powyższą informację o prawach autorskich.

Historia zmian

Wersja	Data	Uwagi
CTFL v4.0.1	15.09.2024	CTFL v4.0.1 – errata
CTFL v4.0	21.04.2023	CTFL v4.0 – nowa wersja
CTFL v3.1.1	1.07.2021	CTFL v3.1.1 – aktualizacja praw autorskich i logo
CTFL v3.1	11.11.2019	CTFL v3.1 – wydanie pielęgnacyjne z niewielkimi aktualizacjami
ISTQB 2018	27.04.2018	CTFL v3.0 – nowa wersja sylabusa
ISTQB 2011	1.04.2011	Aktualizacja sylabusa
ISTQB 2010	30.03.2010	Aktualizacja sylabusa
ISTQB 2007	1.05.2007	Aktualizacja sylabusa
ISTQB 2005	1.07.2005	CTFL v2.0 – pierwsza wersja sylabusa ISTQB poziom podstawowy
ASQF v2.2	07.2003	Sylabus ASQF poziom podstawowy wersja v2.2 „*Lehrplan Grundlagen des Software-testens*”
ISEB v2.0	25.02.1999	Sylabus ISEB *Software Testing Foundation* v2.0

Historia zmian polskiej wersji dokumentu

Wersja	Data	Uwagi
4.0.1.0	14.03.2026	Opublikowanie polskiego tłumaczenia sylabusa w wersji 4.0.1

© International Software Testing Qualifications Board © Polish Quality Board

Spis treści

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board

Podziękowania

Niniejszy dokument został oficjalnie opublikowany przez właściciela produktu / przewodniczącego grupy roboczej Erica Riou du Cosquera w dniu 15.09.2024 r. Został on opracowany przez zespół złożony z członków wspólnych grup roboczych ISTQB® Foundation Level i Agile: Renzo Cerquozzi (wiceprzewodniczący), Wim Decoutere, Jean-François Riverin, Arnika Hryszko, Martin Klonk, Meile Posthuma, Eric Riou du Cosquer (przewodniczący), Adam Roman, Lucjan Stapp, Stephanie Ulrich (wiceprzewodnicząca), Eshraka Zakaria.

Wersja 4.0 niniejszego dokumentu została oficjalnie opublikowana przez Zgromadzenie Ogólne ISTQB® w dniu 21 kwietnia 2023 r. Została ona opracowana przez zespół złożony z członków wspólnych grup roboczych ISTQB® Foundation Level i Agile: Laura Albert, Renzo Cerquozzi (wiceprzewodniczący), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (współprzewodniczący), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (współprzewodniczący), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (wiceprzewodnicząca), Eshraka Zakaria.

Zespół dziękuje Stuartowi Reidowi, Patricii McQuaid i Leanne Howard za przegląd techniczny oraz zespołowi recenzentów i radom krajowym za sugestie i uwagi.

W przeglądzie, komentowaniu i głosowaniu nad niniejszym sylabusem uczestniczyły następujące osoby: Adam Roman, Adam Ścierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Paweł Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radosław Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo i Zsolt Hargitai.

© International Software Testing Qualifications Board © Polish Quality Board

Grupa Robocza ISTQB® ds. Poziomu Podstawowego (wydanie 2018): Klaus Olsen (przewodniczący), Tauhida Parveen (wiceprzewodnicząca), Rex Black (kierownik projektu), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radosław Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery. Główny zespół dziękuje wszystkim radom krajowym za ich sugestie.

Grupa Robocza ISTQB® ds. Poziomu Podstawowego (wydanie 2011): Thomas Müller (przewodniczący), Debra Friedenberg. Główny zespół dziękuje zespołowi recenzentów (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquer, Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) oraz wszystkim radom krajowym za sugestie dotyczące aktualnej wersji sylabusa.

Grupa Robocza ISTQB® ds. Poziomu Podstawowego (wydanie 2010): Thomas Müller (przewodniczący), Rahul Verma, Martin Klonk i Armin Beer. Główny zespół dziękuje zespołowi recenzentów (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) oraz wszystkim radom krajowym za ich sugestie.

Grupa Robocza ISTQB® ds. Poziomu Podstawowego (wydanie 2007): Thomas Müller (przewodniczący), Dorothy Graham, Debra Friedenberg i Erik van Veenendaal. Główny Zespół dziękuje zespołowi recenzentów (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson i Wonil Kwon) oraz wszystkim radom krajowym za ich sugestie.

Grupa Robocza ISTQB® ds. Poziomu Podstawowego (wydanie 2005): Thomas Müller (przewodniczący), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson i Erik van Veenendaal. Główny zespół dziękuje zespołowi recenzentów i wszystkim radom krajowym za ich sugestie.

© International Software Testing Qualifications Board © Polish Quality Board

Wstęp

Cel niniejszego sylabusa

Niniejszy sylabus stanowi podstawę dla międzynarodowej kwalifikacji ISTQB® w zakresie testowania oprogramowania na poziomie podstawowym. ISTQB® udostępnia niniejszy sylabus:

1. Radom krajowym, w celu przetłumaczenia go na język lokalny i akredytacji dostawców szkoleń. Rady krajowe mogą dostosować sylabus do swoich potrzeb językowych i zmodyfikować źródła bibliograficzne, aby dostosować je do lokalnych publikacji.
2. Organom certyfikującym, w celu opracowania pytań egzaminacyjnych w ich lokalnym języku, dostosowanych do celów nauczania określonych w niniejszym sylabusie.
3. Organizatorom szkoleń, w celu opracowania materiałów szkoleniowych i określenia odpowiednich metod nauczania.
4. Kandydatom do certyfikacji, w celu przygotowania się do egzaminu certyfikacyjnego (w ramach szkolenia lub samodzielnie).

Certyfikowany tester poziomu podstawowego w testowaniu oprogramowania

Kwalifikacja poziomu podstawowego jest przeznaczona dla wszystkich osób zajmujących się testowaniem oprogramowania. Obejmuje to osoby pełniące takie role jak testerzy, analitycy testów, inżynierowie testów, konsultanci testów, kierownicy testów, programiści i członkowie zespołów programistycznych. Kwalifikacja poziomu podstawowego jest również odpowiednia dla wszystkich osób, które chcą uzyskać podstawową wiedzę na temat testowania oprogramowania, takich jak kierownicy projektów, kierownicy ds. jakości, właściciele produktów, kierownicy ds. rozwoju oprogramowania, analitycy biznesowi, dyrektorzy IT i konsultanci ds. zarządzania. Posiadacze certyfikatu podstawowego mogą następnie uzyskać kolejne, wyższe kwalifikacje w zakresie testowania oprogramowania.

Ścieżki kariery dla testerów

Program ISTQB® zapewnia wsparcie dla specjalistów ds. testowania na wszystkich etapach ich kariery, oferując szeroką i dogłębną wiedzę. Osoby, które uzyskały certyfikat ISTQB® Certyfikowany Tester Poziom Podstawowy mogą być również zainteresowane poziomem zaawansowanym (analityk testów, techniczny analityk testów, zarządzanie testami), a następnie poziomem eksperckim (zarządzanie testami i doskonalenie procesu testowego). Każdy, kto chce rozwijać umiejętności w zakresie praktyk testowania w zwinnym wytwarzaniu oprogramowania, może rozważyć certyfikaty techniczny tester zwinny lub Agile Test Leadership at Scale. Ścieżka specjalistyczna oferuje sylabusy dotyczące specyficznych podejść do testowania i czynności testowych (np. strategia automatyzacji testów, testowanie AI, testowanie oparte na modelu, testowanie aplikacji mobilnych), które są związane z konkretnymi obszarami testowania (np. testowanie wydajności, użyteczności, akceptacyjne, zabezpieczeń) lub określonych dziedzin przemysłowych (np. testowanie automotive, testowanie gier). Najnowsze informacje na temat programu certyfikacji testerów ISTQB® można znaleźć na stronie www.istqb.org.

© International Software Testing Qualifications Board © Polish Quality Board

Cele biznesowe

W tej sekcji wymieniono cele biznesowe (ang. *Business Outcomes*) osiągane przez kandydata, który uzyskał certyfikat Certyfikowany Tester Poziom Podstawowy.

Certyfikowany tester poziomu podstawowego potrafi...

FL-BO1	Rozumieć, czym jest testowanie i dlaczego jest korzystne
FL-BO2	Rozumieć podstawowe pojęcia związane z testowaniem oprogramowania.
FL-BO3	Określić podejście do testowania i działania, które należy wdrożyć w zależności od kontekstu testowania
FL-BO4	Oceniać i udoskonalać jakość dokumentacji
FL-BO5	Zwiększać skuteczność i wydajność testowania
FL-BO6	Dostosować proces testowania do cyklu życia oprogramowania
FL-BO7	Rozumieć zasady zarządzania testowaniem
FL-BO8	Sporządzać i przekazywać jasne i zrozumiałe raporty o defektach
FL-BO9	Rozumieć czynniki wpływające na priorytety i działania związane z testowaniem
FL-BO10	Pracować w ramach zespołu interdyscyplinarnego
FL-BO11	Rozumieć ryzyka i korzyści związane z automatyzacją testów
FL-BO12	Określać podstawowe umiejętności wymagane do testowania
FL-BO13	Rozumieć wpływ ryzyka na testowanie
FL-BO14	Skutecznie raportować postępy i jakość testów

Cele nauczania oraz poziomy wiedzy

Cele nauczania (ang. *Learning Objectives*) wspierają cele biznesowe i są podstawą do tworzenia pytań egzaminacyjnych w egzaminach poziomu podstawowego. Co do zasady, cała zawartość rozdziałów 1-6 niniejszego sylabusa podlega egzaminowaniu na poziomie K1. Oznacza to, że kandydat może zostać poproszony o rozpoznanie, zapamiętanie lub przypomnienie sobie słowa kluczowego lub pojęcia wspomnianego w dowolnym z sześciu rozdziałów. Szczegółowe poziomy celów nauczania są przedstawione na początku każdego rozdziału sylabusa i sklasyfikowane w następujący sposób:

- K1 — zapamiętać (ang. *remember*),
- K2 — zrozumieć (ang. *understand*),
- K3 — zastosować (ang. *apply*).

Dalsze szczegóły i przykłady celów nauczania podano w załączniku A. Wszystkie terminy wymienione jako słowa kluczowe tuż pod nagłówkami rozdziałów należy zapamiętać (K1), nawet jeśli nie zostały one wyraźnie wymienione w celach nauczania.

Egzamin certyfikacyjny na poziomie podstawowym

Egzamin certyfikacyjny na poziomie podstawowym opiera się na niniejszym sylabusie. Odpowiedzi na pytania egzaminacyjne mogą wymagać wykorzystania materiałów z więcej niż jednej sekcji niniejszego sylabusa. Wszystkie sekcje sylabusa podlegają egzaminowaniu, z wyjątkiem wprowadzenia i załączników. Wymienione w dokumencie normy i książki są dołączone jako materiały referencyjne (rozdział 7), ale ich treść nie podlega egzaminowaniu, poza

© International Software Testing Qualifications Board © Polish Quality Board

tym, co zostało podsumowane w samym sylabusie na podstawie tych norm i książek. Należy zapoznać się z dokumentami „Struktura i zasady egzaminu" oraz „Tabele struktury egzaminów" (ang. *Exam Structure Tables*).

Akredytacja

Rada krajowa ISTQB® może akredytować dostawców szkoleń, których materiały szkoleniowe są zgodne z niniejszym sylabusem. Dostawcy szkoleń powinni uzyskać wytyczne dotyczące akredytacji od rady krajowej lub organu przeprowadzającego akredytację. Akredytowane szkolenie jest uznawane za zgodne z niniejszym sylabusem i może obejmować egzamin ISTQB® w ramach szkolenia. Wytyczne dotyczące akredytacji dla niniejszego sylabusa są zgodne z ogólnymi wytycznymi dotyczącymi akredytacji opublikowanymi przez Grupę Roboczą ds. Zarządzania Procesami i zgodności (ISTQB® Processes Management and Compliance Working Group).

Odniesienia do norm

W sylabusie poziomu podstawowego znajdują się odniesienia do norm (np. IEEE lub ISO). Odniesienia te stanowią ramy (jak w przypadku odniesień do normy ISO 25010 dotyczącej charakterystyk jakościowych) lub źródło dodatkowych informacji dla czytelników, którzy wyrażą zainteresowanie określonym obszarem wiedzy. Treść norm nie podlega egzaminowaniu. Więcej informacji na temat norm znajduje się w rozdziale 7.

Aktualność

Branża oprogramowania zmienia się bardzo szybko. Aby sprostać tym zmianom i zapewnić interesariuszom dostęp do aktualnych i istotnych informacji, grupy robocze ISTQB® utworzyły na stronie internetowej www.istqb.org linki odsyłające do dokumentacji pomocniczej i do zmian w normach. Informacje te nie są przedmiotem egzaminów w ramach programu poziomu podstawowego.

Poziom szczegółowości

Poziom szczegółowości niniejszego sylabusa pozwala na prowadzenie spójnych szkoleń i egzaminów na poziomie międzynarodowym. Aby osiągnąć ten cel, sylabus składa się z:

- ogólnych celów biznesowych opisujących intencje poziomu podstawowego,
- listy słów kluczowych, które studenci muszą być w stanie zapamiętać,
- celów nauczania dla każdego obszaru wiedzy, opisujące efekty uczenia się, które należy osiągnąć,
- opisu kluczowych pojęć, w tym odniesienia do uznanych źródeł.

Treść sylabusa nie stanowi opisu całego obszaru wiedzy dotyczącego testowania oprogramowania; odzwierciedla ona poziom szczegółowości, jaki należy uwzględnić w szkoleniach poziomu podstawowego. Koncentruje się ona na koncepcjach i technikach testowania, które mogą mieć zastosowanie do wszystkich projektów wytwarzania oprogramowania, bez względu na przyjęty model wytwarzania oprogramowania.

© International Software Testing Qualifications Board © Polish Quality Board

Struktura sylabusa

Sylabus składa się z sześciu rozdziałów zawierających treści podlegające egzaminowaniu. Nagłówek najwyższego poziomu każdego rozdziału określa minimalny czas szkolenia dla danego rozdziału. Czas nie jest podany poniżej poziomu rozdziału. W przypadku akredytowanych szkoleń sylabus wymaga minimum 1135 minut (18 godzin i 55 minut) zajęć, rozłożonych na sześć rozdziałów w następujący sposób:

Rozdział 1: Podstawy testowania (180 minut)

- Kandydat poznaje podstawowe zasady związane z testowaniem, powody, dla których testowanie jest wymagane, oraz cele testów.
- Kandydat rozumie proces testowy, główne czynności związane z testowaniem oraz produkty pracy związane z testowaniem.
- Kandydat zna podstawowe umiejętności niezbędne do testowania.

Rozdział 2: Testowanie w całym cyklu życia oprogramowania (130 minut)

- Kandydat poznaje sposoby włączania testowania do różnych podejść do tworzenia oprogramowania.
- Kandydat poznaje koncepcje podejścia „najpierw test” oraz DevOps.
- Kandydat poznaje różne poziomy testów, typy testów oraz testowanie pielęgnacyjne.

Rozdział 3: Testowanie statyczne (80 minut)

- Kandydat poznaje podstawy testowania statycznego, proces przekazywania informacji zwrotnych i proces przeglądu.

Rozdział 4: Analiza i projektowanie testów (390 minut)

- Kandydat poznaje, jak stosować techniki czarnoskrzynkowe, białoskrzynkowe oraz oparte na doświadczeniu w celu wyprowadzenia przypadków testowych z różnych produktów pracy związanych z oprogramowaniem.
- Kandydat poznaje podejście do testowania oparte na współpracy.

Rozdział 5: Zarządzanie czynnościami testowymi (335 minut)

- Kandydat poznaje ogólne zasady planowania testów oraz szacowania wysiłku testowego.
- Kandydat dowiaduje się, w jaki sposób ryzyka mogą wpływać na zakres testów.
- Kandydat uczy się, jak monitorować i nadzorować czynności testowe.
- Kandydat poznaje, w jaki sposób zarządzanie konfiguracją wspiera testowanie.
- Kandydat uczy się, jak zgłaszać defekty w jasny i zrozumiały sposób.

Rozdział 6: Narzędzia testowe (20 minut)

- Kandydat uczy się klasyfikować narzędzia oraz rozumie ryzyka i korzyści związane z automatyzacją testów.

© International Software Testing Qualifications Board © Polish Quality Board

1. Podstawy testowania – 180 minut

Słowa kluczowe

analiza testów, awaria, cel testów, dane testowe, debugowanie, defekt, implementacja testów, jakość, monitorowanie testów, nadzór nad testami, planowanie testów, podstawa testów, podstawowa przyczyna, pokrycie, pomyłka, procedura testowa, proces testowy, projektowanie testów, przedmiot testów, przypadek testowy, śledzenie powiązań, testalia, testowanie, ukończenie testów, walidacja, warunek testowy, weryfikacja, wykonywanie testów, wynik testu, zapewnienie jakości

Cele nauczania dla rozdziału 1

1.1 Co to jest testowanie?

FL-1.1.1 (K1) Wskazać typowe cele testów.

FL-1.1.2 (K2) Odróżniać testowanie od debugowania.

1.2 Dlaczego testowanie jest niezbędne?

FL-1.2.1 (K2) Podać przykłady wskazujące, dlaczego testowanie jest niezbędne.

FL-1.2.2 (K1) Pamiętać, jaka jest relacja między testowaniem a zapewnieniem jakości.

FL-1.2.3 (K2) Odróżniać podstawową przyczynę, pomyłkę, defekt i awarię.

1.3 Zasady testowania

FL-1.3.1 (K2) Objaśnić siedem zasad testowania.

1.4 Czynności testowe, testalia i role związane z testami

FL-1.4.1 (K2) Wyjaśnić poszczególne czynności testowe i powiązane z nimi zadania.

FL-1.4.2 (K2) Wyjaśnić wpływ kontekstu na proces testowy.

FL-1.4.3 (K2) Rozróżniać testalia wspomagające czynności testowe.

FL-1.4.4 (K2) Wyjaśnić korzyści wynikające ze śledzenia powiązań.

FL-1.4.5 (K2) Porównać poszczególne role występujące w testowaniu.

1.5 Niezbędne umiejętności i dobre praktyki w dziedzinie testowania

FL-1.5.1 (K2) Podać przykłady ogólnych umiejętności wymaganych w testowaniu.

FL-1.5.2 (K1) Pamiętać, jakie są zalety podejścia „cały zespół”.

FL-1.5.3 (K2) Odróżniać korzyści i wady niezależności testowania.

© International Software Testing Qualifications Board © Polish Quality Board

1.1. Co to jest testowanie?

Systemy oprogramowania są integralną częścią naszego codziennego życia. Większość ludzi miała do czynienia z oprogramowaniem, które nie działało zgodnie z oczekiwaniami. Nieprawidłowo działające oprogramowanie może prowadzić do wielu problemów, w tym utraty pieniędzy, czasu lub reputacji firmy, a w skrajnych przypadkach nawet utraty zdrowia lub życia. Testowanie oprogramowania pozwala ocenić jego jakość i pomaga zmniejszyć ryzyko awarii podczas użytkowania.

Testowanie oprogramowania to zestaw czynności mających na celu wykrycie defektów i ocenę jakości produktów pracy. Produkty pracy poddawane takim testom nazywane są przedmiotem testów. Powszechnym błędnym przekonaniem na temat testowania jest to, że polega ono wyłącznie na wykonywaniu testów (tj. uruchamianiu oprogramowania i sprawdzaniu wyników testów). Jednak testowanie oprogramowania obejmuje również inne czynności i musi być dostosowane do cyklu życia oprogramowania (patrz rozdział 2).

Kolejnym powszechnym błędnym przekonaniem dotyczącym testowania jest to, że skupia się ono wyłącznie na weryfikacji przedmiotu testów. Testowanie obejmuje weryfikację, tj. sprawdzenie, czy system spełnia określone wymagania, a także walidację, tj. sprawdzanie, czy system spełnia potrzeby użytkowników i innych interesariuszy w swoim środowisku operacyjnym.

Testowanie może być dynamiczne lub statyczne. Testowanie dynamiczne wiąże się z uruchamianiem oprogramowania, natomiast testowanie statyczne nie. Testowanie statyczne obejmuje przeglądy (patrz rozdział 3) i analizę statyczną. Testowanie dynamiczne wykorzystuje różne techniki testowania i podejścia do testów w celu wyprowadzania przypadków testowych (patrz rozdział 4).

Testowanie nie jest wyłącznie działaniem technicznym. Wymaga ono również odpowiedniego planowania, zarządzania, szacowania, monitorowania i nadzoru (patrz rozdział 5). Testerzy korzystają z narzędzi (patrz rozdział 6), ale należy pamiętać, że testowanie jest działaniem intelektualnym, wymagającym od testerów specjalistycznej wiedzy, umiejętności analitycznych oraz krytycznego i systemowego myślenia (Myers 2011, Roman 2018).

Więcej informacji na temat koncepcji testowania oprogramowania zawiera norma ISO/IEC/IEEE 29119-1.

1.1.1. Cele testów

Typowe cele testów to:

- ocena produktów pracy, takich jak wymagania, historyjki użytkowników, projekty i kod,
- wywoływanie awarii i wykrywanie defektów,
- zapewnienie wymaganego pokrycia przedmiotu testów,
- zmniejszenie poziomu ryzyka związanego z nieodpowiednią jakością oprogramowania,
- weryfikacja, czy wyspecyfikowane wymagania zostały spełnione,
- dostarczanie informacji interesariuszom, aby umożliwić im podejmowanie świadomych decyzji,
- budowanie zaufania do jakości przedmiotu testów,
- sprawdzenie, czy przedmiot testów jest kompletny i działa zgodnie z oczekiwaniami interesariuszy.

© International Software Testing Qualifications Board © Polish Quality Board

Cele testów mogą się różnić w zależności od kontekstu, który obejmuje testowany produkt, poziom testów, ryzyka, stosowany cykl wytwarzania oprogramowania (SDLC) oraz czynniki związane z kontekstem biznesowym, np. struktura korporacyjna, kwestie konkurencyjności lub czas wprowadzenia produktu na rynek.

1.1.2. Testowanie a debugowanie

Testowanie i debugowanie to odrębne czynności. Testowanie może wywołać awarie spowodowane defektami w oprogramowaniu (testowanie dynamiczne) lub bezpośrednio wykryć defekty przedmiotu testów (testowanie statyczne).

Gdy testowanie dynamiczne (patrz rozdział 4) powoduje awarię, debugowanie polega na znalezieniu przyczyn tej awarii (defektów), analizie tych przyczyn i ich wyeliminowaniu. Typowy proces debugowania w tym przypadku obejmuje:

- odtworzenie awarii,
- diagnozę (znalezienie defektu),
- naprawę (usunięcie defektu).

Wykonywane następnie testy potwierdzające sprawdzają, czy poprawki rozwiązały problem. Najlepiej, aby testy potwierdzające były przeprowadzane przez tę samą osobę, która wykonywała wcześniejsze testy. Można również przeprowadzić testy regresji, aby sprawdzić, czy poprawki nie powodują awarii w innych częściach przedmiotu testów (więcej informacji na temat testowania potwierdzającego i testowania regresji znajduje się w sekcji 2.2.3).

Gdy testy statyczne wykryją defekt, debugowanie polega na jego usunięciu. Nie ma potrzeby odtwarzania ani diagnozowania, ponieważ testy statyczne wykrywają defekty bezpośrednio i nie mogą powodować awarii (patrz rozdział 3).

1.2. Dlaczego testowanie jest niezbędne?

Testowanie jako forma kontroli jakości pomaga w osiągnięciu uzgodnionych celów testów w ramach ustalonego zakresu, czasu, jakości i ograniczeń budżetowych. Wkład testowania w sukces przedsięwzięcia nie powinien ograniczać się do działań zespołu testowego. Każdy interesariusz może wykorzystać swoje umiejętności testowania, aby zwiększyć szanse na sukces projektu. Testowanie modułów, systemów i powiązanych produktów pracy (np. dokumentacji) pomaga w identyfikacji defektów w oprogramowaniu.

1.2.1. Znaczenie testowania dla powodzenia projektu

Testowanie stanowi opłacalny sposób wykrywania defektów. Defekty te można następnie usunąć (poprzez debugowanie – czynność niezwiązaną z testowaniem), więc testowanie pośrednio przyczynia się do poprawy jakości przedmiotów testów.

Testowanie stanowi środek bezpośredniej oceny jakości przedmiotu testów na różnych etapach cyklu życia oprogramowania. Środki te są wykorzystywane w ramach szerszej działalności związanej z zarządzaniem projektami, przyczyniając się do podejmowania decyzji o przejściu do kolejnego etapu cyklu wytwarzania oprogramowania, takiego jak decyzja o wydaniu.

Testowanie zapewnia użytkownikom pośrednią reprezentację w projekcie wytwórczym. Testerzy dbają o to, aby rozumiane przez nich potrzeby użytkowników były brane pod uwagę w całym cyklu

© International Software Testing Qualifications Board © Polish Quality Board

wytwarzania. Alternatywą jest zaangażowanie reprezentatywnej grupy użytkowników w projekt wytwórczy, co zazwyczaj nie jest możliwe ze względu na wysokie koszty i brak odpowiednich użytkowników.

Testowanie może być również wymagane w celu spełnienia wymogów umów, przepisów prawa lub zapewnienia zgodności z normami regulacyjnymi.

1.2.2. Testowanie a zapewnienie jakości (QA)

Chociaż terminy „testowanie" i „zapewnienie jakości" (QA – ang. *quality assurance*) są często używane zamiennie, to jednak testowanie i QA nie są tym samym.

Testowanie jest podejściem korygującym i zorientowanym na produkt, koncentrującym się na działaniach wspierających osiągnięcie odpowiedniego poziomu jakości. Testowanie jest główną formą kontroli jakości. Inne formy tej kontroli obejmują metody formalne (weryfikacja modelu czy formalne dowodzenie poprawności), symulację i prototypowanie.

QA jest podejściem prewencyjnym i zorientowanym na procesy, koncentrującym się na ich wdrażaniu i doskonaleniu. Opiera się na założeniu, że prawidłowe stosowanie dobrego procesu pozwoli uzyskać dobry produkt. QA ma zastosowanie zarówno do procesów wytwarzania, jak i testowania, a odpowiedzialność za nie spoczywa na wszystkich uczestnikach projektu.

Wyniki testów są wykorzystywane zarówno przez QA, jak i przez testowanie. W testowaniu służą one do usuwania defektów, natomiast w QA dostarczają informacji zwrotnych na temat skuteczności procesów wytwarzania i testowania.

1.2.3. Pomyłki, defekty, awarie i podstawowe przyczyny

Ludzie popełniają błędy, które powodują defekty (usterki, bugi), a te z kolei mogą prowadzić do awarii. Powody popełniania błędów przez człowieka to np. presja czasu, złożoność produktów pracy, procesów, infrastruktury lub interakcji, zmęczenie lub brak odpowiedniego przeszkolenia.

Defekty można znaleźć w dokumentacji (np. w specyfikacja wymagań lub skrypcie testowym), w kodzie źródłowym lub w pomocniczym produkcie pracy (np. plik kompilacji). Niewykrycie defektów w produktach pracy wytworzonych we wcześniejszych etapach cyklu wytwarzania oprogramowania często prowadzi do wadliwych produktów pracy w późniejszych etapach tego cyklu. Wykonanie kodu zawierającego defekt może spowodować, że system nie wykona tego, co powinien, lub wykona coś, czego nie powinien, powodując awarię. Niektóre defekty zawsze powodują awarię po wykonaniu, podczas gdy inne powodują awarię tylko w określonych okolicznościach, a niektóre mogą nigdy nie powodować awarii.

Awarie mogą być spowodowane nie tylko błędami i defektami, ale też warunkami środowiskowymi, na przykład, gdy promieniowanie lub działanie pola elektromagnetycznego powoduje nieprawidłowe działanie oprogramowania wbudowanego (ang. *firmware*).

Podstawowa przyczyna to podstawowy, źródłowy powód wystąpienia problemu (np. sytuacja prowadząca do pomyłki). Podstawowe przyczyny są identyfikowane w toku tzw. analizy przyczyny podstawowej (ang. *root cause analysis*), przeprowadzanej zwykle po wystąpieniu awarii lub wykryciu defektu. Uważa się, że wyeliminowanie podstawowej przyczyny (ang. *root cause*) może zapobiec lub zmniejszyć częstotliwość dalszych podobnych awarii lub defektów.

© International Software Testing Qualifications Board © Polish Quality Board

1.3. Zasady testowania

Na przestrzeni lat zaproponowano szereg zasad testowania, które stanowią ogólne wytyczne mające zastosowanie do wszystkich testów. Niniejszy sylabus opisuje siedem takich zasad.

1. Testowanie ujawnia defekty, ale nie może dowieść ich braku. Testowanie może wykazać istnienie defektów w przedmiocie testów, ale nie może udowodnić, że jest on od nich wolny (Buxton 1970). Testowanie zmniejsza prawdopodobieństwo, że defekty pozostaną niewykryte w przedmiocie testów, ale niewykrycie żadnych defektów nie jest dowodem poprawności przedmiotu testów.

2. Testowanie gruntowne jest niemożliwe. Testowanie wszystkiego jest niewykonalne, z wyjątkiem trywialnych przypadków (Manna 1978). Zamiast przeprowadzać wyczerpujące testy, należy skupić się na technikach testowania (patrz rozdział 4), priorytetyzacji przypadków testowych (patrz sekcja 5.1.5) i testowaniu opartym na ryzyku (patrz podrozdział 5.2).

3. Wczesne testowanie oszczędza czas i pieniądze. Defekty usunięte na wczesnym etapie procesu nie spowodują kolejnych defektów w pochodnych produktach pracy. Koszt jakości zmniejszy się, ponieważ w późniejszym etapie cyklu wytwarzania oprogramowania wystąpi mniej awarii (Boehm 1981). Aby wcześnie wykryć defekty, należy jak najwcześniej rozpocząć zarówno testowanie statyczne (patrz rozdział 3), jak i testowanie dynamiczne (patrz rozdział 4).

4. Defekty mogą się kumulować. Niewielka liczba modułów systemu zawiera zazwyczaj większość wykrytych defektów lub jest odpowiedzialna za większość awarii operacyjnych (Enders 1975). Zjawisko to ilustruje zasadę Pareto. Przewidywane skupiska defektów oraz rzeczywiste skupiska defektów zaobserwowane podczas testowania lub eksploatacji stanowią ważny wkład w testowanie oparte na ryzyku (patrz podrozdział 5.2).

5. Testy ulegają zużyciu. Jeśli te same testy są powtarzane wiele razy, stają się coraz mniej skuteczne w wykrywaniu nowych defektów (Beizer 1990). Aby przezwyciężyć ten efekt, może być konieczna modyfikacja istniejących testów i danych testowych oraz napisanie nowych testów. Jednak w niektórych przypadkach powtarzanie tych samych testów może być korzystne, np. w automatycznych testach regresji (patrz sekcja 2.2.3).

6. Testowanie zależy od kontekstu. Nie ma jednego uniwersalnego podejścia do testowania. Testowanie przebiega inaczej w różnych kontekstach (Kaner 2011).

7. Przekonanie o braku defektów jest błędem. Błędem jest oczekiwanie, że sama weryfikacja zapewni sukces systemu. Dokładne przetestowanie wszystkich wyspecyfikowanych wymagań i usunięcie wszystkich znalezionych defektów może nadal skutkować powstaniem systemu, który nie spełnia potrzeb i oczekiwań użytkowników, nie pomaga w osiągnięciu celów biznesowych klienta i jest gorszy od innych konkurencyjnych systemów. Oprócz weryfikacji należy zatem również przeprowadzać walidację (Boehm 1981).

1.4. Czynności testowe, testalia i role związane z testami

Testowanie zależy od kontekstu, ale istnieją typowe grupy czynności testowych, których pominięcie zmniejsza szanse osiągnięcia celów testów. Te grupy czynności testowych tworzą proces testowy. Dobór procesu testowego do danej sytuacji zależy od wielu czynników. To, jakie czynności testowe są uwzględnione w tym procesie testowym, w jaki sposób są one wdrażane

© International Software Testing Qualifications Board © Polish Quality Board

i kiedy mają miejsce, jest zazwyczaj ustalane w ramach planowania testów z uwzględnieniem kontekstu (patrz podrozdział 5.1).

Poniżej opisano ogólne aspekty tego procesu testowego w odniesieniu do czynności i zadań testowych, wpływu kontekstu, testaliów, śledzenia powiązań między podstawą testów a testaliami oraz ról występujących w testowaniu.

Więcej informacji na temat procesów testowych zawiera norma ISO/IEC/IEEE 29119-2.

1.4.1. Czynności i zadania testowe

W procesie testowym wyróżnia się główne grupy czynności opisane poniżej. Chociaż wiele z tych czynności może sprawiać wrażenie logicznie uszeregowanych, często są one realizowane metodą iteracyjną lub równolegle. Ponadto zwykle wymagają one dostosowania do potrzeb konkretnego systemu i projektu.

Planowanie testów polega na zdefiniowaniu celów testów, a następnie wybraniu podejścia, które najlepiej pozwoli osiągnąć te cele w ramach ograniczeń narzuconych przez ogólny kontekst. Planowanie testów zostało szczegółowo omówione w podrozdziale 5.1.

Monitorowanie testów i nadzór nad testami. Monitorowanie testów obejmuje ciągłą kontrolę wszystkich czynności testowych oraz porównanie rzeczywistych postępów z planem. Nadzór nad testami obejmuje podejmowanie działań niezbędnych do osiągnięcia celów testów. Monitorowanie testów i nadzór nad testami zostały szczegółowo omówione w podrozdziale 5.3.

Analiza testów obejmuje analizę podstawy testów w celu zidentyfikowania cech, które można przetestować oraz zdefiniowanie i spriorytetyzowanie powiązanych z nimi warunków testowych, biorąc pod uwagę istniejące ryzyka (patrz podrozdział 5.2). Analizuje się również podstawę testów i przedmiot testów w celu oceny ich testowalności oraz zidentyfikowania ewentualnych defektów. Analiza testów jest często wspierana przez zastosowanie technik testowania (patrz rozdział 4). Analiza testów odpowiada na pytanie: „co testować?” w kategoriach mierzalnych kryteriów pokrycia.

Projektowanie testów przekształca warunki testowe w przypadki testowe i inne testalia (np. karty opisu testów). Czynność ta często wiąże się z identyfikacją elementów pokrycia, które służą jako wskazówki do określenia danych wejściowych dla przypadków testowych. Do wsparcia tej czynności można wykorzystać techniki testowania (patrz rozdział 4). Projektowanie testów obejmuje również określenie wymagań dotyczących danych testowych, zaprojektowanie środowiska testowego oraz identyfikację niezbędnej infrastruktury i narzędzi. Projektowanie testów odpowiada na pytanie: „jak testować?”.

Implementacja testów obejmuje tworzenie lub pozyskiwanie testaliów niezbędnych do wykonania testów (np. danych testowych), organizację przypadków testowych w procedury testowe i zestawy testowe, tworzenie manualnych i automatycznych skryptów testowych. Procedury testowe są szeregowane według priorytetów i układane w harmonogramie wykonywania testów w celu zapewnienia efektywnego wykonywania testów (patrz sekcja 5.1.5). Środowisko testowe jest budowane i weryfikowane pod kątem prawidłowej konfiguracji.

Wykonywanie testów obejmuje przeprowadzenie testów zgodnie z harmonogramem wykonania testów (czyli wykonywanie przebiegów testów). Wykonywanie testów może być manualne lub automatyczne i może przybierać różne formy, np. testowania ciągłego lub sesji testowania

© International Software Testing Qualifications Board © Polish Quality Board

w parach. Rzeczywiste wyniki testów są porównywane z wynikami oczekiwanymi. Wyniki testów są rejestrowane. Anomalie są analizowane w celu zidentyfikowania ich prawdopodobnych przyczyn. Analiza ta pozwala zgłaszać anomalie na podstawie zaobserwowanych awarii (patrz podrozdział 5.5).

Ukończenie testów następuje zazwyczaj w momencie osiągnięcia kamieni milowych projektu (np. wydanie, koniec iteracji, ukończenie testów danego poziomu). W przypadku nierozwiązanych defektów tworzone są żądania zmian lub pozycje w backlogu produktu. Wszelkie testalia, które mogą być przydatne w przyszłości, są identyfikowane i archiwizowane lub przekazywane odpowiednim zespołom. Środowisko testowe jest zamykane w uzgodnionym stanie. Czynności testowe są analizowane w celu sformułowania wniosków i zidentyfikowania udoskonaleń dla przyszłych iteracji, wydań lub projektów (patrz sekcja 2.1.6). Sumaryczny raport z testów jest tworzony i przekazywany interesariuszom.

1.4.2. Proces testowy w kontekście

Testowanie nie jest przeprowadzane w izolacji. Czynności testowe są integralną częścią procesów wytwarzania oprogramowania realizowanych w ramach organizacji. Testowanie jest również finansowane przez interesariuszy, a jego ostatecznym celem jest pomoc w zaspokojeniu ich potrzeb biznesowych. Dlatego sposób przeprowadzania testów będzie zależał od wielu czynników kontekstowych, w tym:

- interesariuszy (ich potrzeb, oczekiwań, wymagań, chęci współpracy itp.),
- członków zespołu (ich umiejętności, wiedzy, poziomu doświadczenia, dostępności, potrzeb szkoleniowych itp.),
- dziedziny biznesowej (krytyczność przedmiotu testów, zidentyfikowane ryzyka, potrzeby rynku, szczegółowe regulacje prawne itp.),
- czynników technicznych (rodzaj oprogramowania, architektura produktu, zastosowana technologia itp.),
- ograniczeń projektu (zakres, czas, budżet, zasoby itp.),
- czynników organizacyjnych (struktura organizacyjna, istniejące polityki, stosowane praktyki itp.),
- cyklu życia oprogramowania (praktyki inżynieryjne, metody rozwoju itp.),
- narzędzi (dostępność, użyteczność, zgodność itp.).

Czynniki te będą miały wpływ na wiele kwestii związanych z testowaniem, w tym na strategię testów, stosowane techniki testowania, stopień automatyzacji testów, wymagany poziom pokrycia, poziom szczegółowości dokumentacji testowej, raportowanie testów itp.

1.4.3. Testalia

Testalia powstają jako produkty pracy czynności testowych opisanych w sekcji 1.4.1. Istnieją znaczne różnice w sposobie, w jaki różne organizacje tworzą, nazywają, organizują i zarządzają swoimi testaliami. Właściwe zarządzanie konfiguracją (patrz podrozdział 5.4) zapewnia spójność i integralność testaliów. Poniższa lista zawiera wybrane przykłady testaliów (produktów pracy).

- **Produkty pracy związane z planowaniem testów** obejmują: plan testów, harmonogram testów, rejestr ryzyk, kryteria wejścia i kryteria wyjścia (patrz podrozdział 5.1). Rejestr ryzyk to lista ryzyk wraz z prawdopodobieństwem, wpływem i sposobem łagodzenia tych ryzyk

© International Software Testing Qualifications Board © Polish Quality Board

(patrz podrozdział 5.2). Harmonogram testów, rejestr ryzyk, kryteria wejścia i kryteria wyjścia często są częścią planu testów.

- **Produkty pracy związane z monitorowaniem testów i nadzorem nad testami** obejmują: raporty o postępie testów (patrz sekcja 5.3.2), dokumentację dyrektyw nadzoru (patrz podrozdział 5.3) oraz informacje o ryzykach (patrz podrozdział 5.2).
- **Produkty pracy związane z analizą testów** obejmują spriorytetyzowane warunki testowe (np. kryteria akceptacji, patrz sekcja 4.5.2) oraz raporty o defektach w podstawie testów (jeśli nie zostały one natychmiast usunięte).
- **Produkty pracy związane z projektowaniem testów** obejmują spriorytetyzowane przypadki testowe, karty opisu testów, elementy pokrycia, wymagania dotyczące danych testowych oraz wymagania dotyczące środowiska testowego.
- **Produkty pracy związane z implementacją testów** obejmują procedury testowe, manualne i zautomatyzowane skrypty testowe, zestawy testowe, dane testowe, harmonogram wykonywania testów oraz elementy środowiska testowego. Przykłady elementów środowiska testowego obejmują zaślepki, sterowniki, symulatory i wirtualizację usług.
- **Produkty pracy związane z wykonywaniem testów** obejmują dzienniki testów i raporty o defektach (patrz podrozdział 5.5).
- **Produkty pracy związane z ukończeniem testów** obejmują sumaryczny raport z testów (patrz sekcja 5.3.2), listę działań mających na celu doskonalenie kolejnych projektów lub iteracji, udokumentowane wnioski oraz żądania zmian (np. jako elementy backlogu produktu).

1.4.4. Śledzenie powiązań między podstawą testów a testaliami

Wdrożenie skutecznego monitorowania testów i nadzoru nad testami wymaga ustanowienia i utrzymywania przez cały czas trwania projektu testowego mechanizmu śledzenia powiązań pomiędzy elementami podstawy testów a testaliami (np. warunkami testowymi, ryzykami, przypadkami testowymi), wynikami testów i defektami.

Dokładne śledzenie powiązań wspiera ocenę pokrycia, dlatego bardzo przydatne jest zdefiniowanie mierzalnych kryteriów pokrycia w podstawie testów. Kryteria pokrycia mogą funkcjonować jako kluczowe wskaźniki wydajności (ang. *key performance indicators* – KPI), które pokazują, w jakim stopniu osiągnięto cele testów (patrz sekcja 1.1.1). Na przykład:

- Śledzenie powiązań między przypadkami testowymi a wymaganiami pozwala zweryfikować, czy wymagania zostały pokryte przypadkami testowymi.
- Śledzenie powiązań między wynikami testów a ryzykami umożliwia ocenę poziomu ryzyka rezydualnego (resztkowego) w przedmiocie testów.

Oprócz oceny pokrycia, śledzenie powiązań umożliwia określenie wpływu zmian, ułatwia audyty i pomaga spełnić kryteria zarządzania IT. Śledzenie powiązań ułatwia rozumienie raportów o postępie testów i sumarycznych raportów z testów dzięki uwzględnieniu statusu elementów podstawy testów. To z kolei pomaga w przekazywaniu interesariuszom informacji na temat technicznych aspektów testowania w zrozumiały sposób. Śledzenie powiązań dostarcza ponadto informacji potrzebnych do oceny jakości produktu, wydajności procesów i postępów w realizacji projektu w odniesieniu do celów biznesowych.

© International Software Testing Qualifications Board © Polish Quality Board

1.4.5. Role w procesie testowania

W niniejszym sylabusie omówiono dwie główne role w testowaniu: rolę związaną z zarządzaniem testami i rolę związaną z testowaniem. Czynności i zadania przypisane do tych dwóch ról zależą od takich czynników jak: kontekst projektu i produktu, umiejętności osób pełniących te role oraz specyfika organizacji.

Rola związana z zarządzaniem testami obejmuje ogólną odpowiedzialność za proces testowy, zespół testowy i kierowanie czynnościami testowymi. Rola ta koncentruje się głównie na działaniach związanych z planowaniem testów, monitorowaniem testów, nadzorem nad testami i ukończeniem testów. Sposób wykonywania tej roli różni się w zależności od kontekstu. Na przykład w ramach zwinnego wytwarzania oprogramowania niektóre zadania związane z zarządzaniem testami mogą być wykonywane przez zespół zwinny. Zadania obejmujące wiele zespołów lub całą organizację mogą być wykonywane przez kierowników testów spoza zespołu tworzącego oprogramowania.

Rola związana z testowaniem obejmuje ogólną odpowiedzialność za inżynieryjny (techniczny) aspekt testowania. Rola ta koncentruje się głównie na działaniach związanych z analizą testów, projektowaniem testów, implementacją testów i wykonywaniem testów.

Różne osoby mogą pełnić powyższe role w różnym czasie. Na przykład rolę związaną z zarządzaniem testami może pełnić kierownik zespołu, kierownik ds. testów, kierownik ds. rozwoju itp. Możliwe jest również, aby jedna osoba pełniła jednocześnie obie role.

1.5. Niezbędne umiejętności i dobre praktyki w dziedzinie testowania

Umiejętność to zdolność do prawidłowego wykonywania czegoś, wynikająca z wiedzy, praktyki i predyspozycji danej osoby. Dobrzy testerzy powinni posiadać pewne podstawowe umiejętności, aby dobrze wykonywać swoją pracę. Powinni być skutecznymi graczami zespołowymi i powinni być w stanie wykonywać testy na różnych poziomach niezależności.

1.5.1. Ogólne umiejętności wymagane w związku z testowaniem

Poniższe umiejętności, choć ogólne, są szczególnie istotne dla testerów:

- wiedza w zakresie testowania (zwiększa skuteczność testowania, np. poprzez stosowanie technik testowania),
- staranność, ostrożność, ciekawość, dbałość o szczegóły, metodyczność (ułatwia identyfikację defektów, zwłaszcza tych trudnych do wykrycia),
- umiejętności komunikacyjne, aktywne słuchanie, umiejętność pracy w zespole (pozwala na skuteczną interakcję ze wszystkimi interesariuszami, efektywne przekazywanie informacji, bycie zrozumiałym oraz sprawne zgłaszanie i omawianie defektów),
- myślenie analityczne, myślenie krytyczne, kreatywność (zwiększa skuteczność testowania),
- wiedza techniczna (zwiększa wydajność testowania, np. poprzez stosowanie odpowiednich narzędzi testowych),
- wiedza dziedzinowa (pozwala zrozumieć użytkowników oraz przedstawicieli jednostek biznesowych i efektywnie komunikować się z nimi).

© International Software Testing Qualifications Board © Polish Quality Board

Testerzy są często posłańcami złych wieści. Obwinianie osoby przynoszącej złe wiadomości jest powszechną cechą ludzką. Dlatego umiejętności komunikacyjne są tak ważne dla testerów. Przekazywanie wyników testów może być postrzegane jako krytyka produktu i jego autora. Efekt potwierdzenia (ang. *confirmation bias*) może utrudniać akceptację informacji, które są sprzeczne z aktualnymi przekonaniami. Niektórzy mogą postrzegać testowanie jako działanie destrukcyjne, mimo że w znacznym stopniu przyczynia się ono do sukcesu projektu i jakości produktu. Aby uniknąć powyższych problemów, informacje o defektach i awariach powinny być przekazywane w konstruktywny sposób.

1.5.2. Podejście „cały zespół”

Jedną z ważnych umiejętności testera jest zdolność do efektywnej pracy w zespole i pozytywnego przyczyniania się do realizacji celów zespołu. Na tej umiejętności opiera się podejście „cały zespół” (ang. *whole-team approach*) – praktyka wywodząca się z programowania ekstremalnego (patrz podrozdział 2.1).

W podejściu „cały zespół” każdy członek zespołu posiadający niezbędną wiedzę i umiejętności może wykonywać dowolne zadanie, a odpowiedzialność za jakość spoczywa w równym stopniu na wszystkich. Członkowie zespołu dzielą tę samą przestrzeń roboczą (fizyczną lub wirtualną), ponieważ wspólna lokalizacja ułatwia komunikację i interakcję. Podejście „cały zespół” poprawia dynamikę zespołu, usprawnia komunikację i współpracę w zespole oraz tworzy synergię, umożliwiając wykorzystanie różnych zestawów umiejętności w zespole z korzyścią dla projektu.

Testerzy ściśle współpracują z innymi członkami zespołu, aby zapewnić osiągnięcie wymaganego poziomu jakości. Obejmuje to również współpracę z przedstawicielami biznesowymi w celu tworzenia testów akceptacyjnych oraz współpracę z programistami w celu uzgodnienia strategii testów i podjęcia decyzji dotyczących automatyzacji testów. Testerzy mogą w ten sposób przekazywać wiedzę na temat testowania innym członkom zespołu i wpływać na proces wytwarzania produktu.

W pewnych przypadkach podejście „cały zespół” nie jest odpowiednie. Na przykład w ramach wytwarzania systemów krytycznych ze względów bezpieczeństwa, może być wymagany wysoki poziom niezależności testów.

1.5.3. Niezależność testowania

Pewien stopień niezależności sprawia, że tester jest bardziej skuteczny w wykrywaniu defektów ze względu na różnice między błędami poznawczymi (ang. *cognitive bias*) autora i testera (por. Salman 1995). Niezależność nie zastępuje jednak znajomości produktu, np. programiści mogą skutecznie wykrywać wiele defektów we własnym kodzie.

Produkty pracy mogą być testowane przez ich autora (brak niezależności), przez współpracowników autora z tego samego zespołu (pewna niezależność), przez testerów spoza zespołu autora, ale z tej samej organizacji (wysoka niezależność) lub przez testerów spoza organizacji (bardzo wysoka niezależność). W przypadku większości projektów zazwyczaj najlepiej jest przeprowadzać testy z wykorzystaniem wielu poziomów niezależności (np. programiści przeprowadzają testowanie modułowe i testowanie integracji modułów, zespół testowy – testowanie systemowe i testowanie integracji systemów, a przedstawiciele biznesowi – testowanie akceptacyjne).

© International Software Testing Qualifications Board © Polish Quality Board

Główną zaletą niezależności testowania jest to, że niezależni testerzy mogą dostrzegać inne rodzaje awarii i defektów niż programiści ze względu na swoje odmienne doświadczenie, perspektywę techniczną i błędy poznawcze. Ponadto niezależny tester może weryfikować, kwestionować lub obalać założenia poczynione przez interesariuszy podczas specyfikacji i implementowania systemu.

Niezależność testowania ma również pewne wady. Niezależni testerzy mogą być odizolowani od zespołu programistów, co może prowadzić do braku współpracy, problemów z komunikacją lub nawet konfliktów z zespołem programistów. Programiści mogą utracić poczucie odpowiedzialności za jakość. Niezależni testerzy mogą być postrzegani jako wąskie gardło lub być obwiniani za opóźnienia w wydaniu produktu.

© International Software Testing Qualifications Board © Polish Quality Board

2. Testowanie w cyklu wytwarzania oprogramowania – 130 minut

Słowa kluczowe

poziom testów, przedmiot testów, przesunięcie w lewo, testowanie akceptacyjne, testowanie białoskrzynkowe, testowanie czarnoskrzynkowe, testowanie funkcjonalne, testowanie integracji modułów, testowanie integracji systemów, testowanie integracyjne, testowanie modułowe, testowanie niefunkcjonalne, testowanie pielęgnacyjne, testowanie potwierdzające, testowanie regresji, testowanie systemowe, typ testów

Cele nauczania dla rozdziału 2

2.1 Testowanie w kontekście cyklu wytwarzania oprogramowania

FL-2.1.1 (K2) Wyjaśnić wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie.

FL-2.1.2 (K1) Pamiętać dobre praktyki testowania mające zastosowanie do wszystkich modeli cyklu wytwarzania oprogramowania.

FL-2.1.3 (K1) Podać przykłady podejść typu „najpierw test” w kontekście wytwarzania oprogramowania.

FL-2.1.4 (K2) Podsumować, w jaki sposób metodyka DevOps może wpłynąć na testowanie.

FL-2.1.5 (K2) Wyjaśnić, na czym polega przesunięcie w lewo.

FL-2.1.6 (K2) Wyjaśnić, w jaki sposób retrospektywy mogą posłużyć jako mechanizmy doskonalenia procesów.

2.2 Poziomy testów i typy testów

FL-2.2.1 (K2) Rozróżniać poszczególne poziomy testów.

FL-2.2.2 (K2) Rozróżniać poszczególne typy testów.

FL-2.2.3 (K2) Odróżniać testowanie potwierdzające od testowania regresji.

2.3 Testowanie pielęgnacyjne

FL-2.3.1 (K2) Podsumować testowanie pielęgnacyjne i zdarzenia je wyzwalające.

© International Software Testing Qualifications Board © Polish Quality Board

2.1. Testowanie w kontekście modelu cyklu wytwarzania oprogramowania

Model cyklu wytwarzania oprogramowania jest abstrakcyjnym, wysokopoziomowym przedstawieniem procesu wytwarzania oprogramowania. Model ten określa, w jaki sposób różne fazy rozwoju i rodzaje czynności wykonywanych w ramach procesu wytwarzania oprogramowania odnoszą się do siebie zarówno pod względem logicznym, jak i chronologicznym. Przykłady obejmują: sekwencyjne modele wytwarzania oprogramowania (np. model kaskadowy, model V), iteracyjne modele wytwarzania oprogramowania (np. model spiralny, prototypowanie) oraz przyrostowe modele wytwarzania oprogramowania (np. *Unified Process*).

Niektóre czynności w ramach procesów wytwarzania oprogramowania można również opisać za pomocą bardziej szczegółowych metod wytwarzania oprogramowania i praktyk zwinnych. Przykłady obejmują: wytwarzanie sterowane testami akceptacyjnymi (ang. *acceptance test-driven development*, ATDD), wytwarzanie sterowane zachowaniem (ang. *behavior-driven development*, BDD), projektowanie oparte na domenie (ang. *domain-driven design*, DDD), programowanie ekstremalne (ang. *eXtreme Programming*, XP), wytwarzanie oparte na cechach (ang. *feature-driven development*, FDD), Kanban, Lean IT, Scrum oraz wytwarzanie sterowane testami (ang. *test-driven development*, TDD).

2.1.1. Wpływ cyklu wytwarzania oprogramowania na testowanie

Warunkiem powodzenia procesu testowego jest jego dostosowanie do przyjętego cyklu wytwarzania oprogramowania. Wybór modelu cyklu wytwarzania oprogramowania ma wpływ na:

- zakres i harmonogram działań testowych (np. poziomy testów i typy testów),
- poziom szczegółowości dokumentacji testowej,
- wybór technik testowania i podejścia do testów,
- zakres automatyzacji testów,
- rolę i obowiązki testera.

W sekwencyjnych modelach wytwarzania oprogramowania, w początkowych fazach procesu, testerzy zazwyczaj uczestniczą w przeglądach wymagań, analizie testów i projektowaniu testów. Wykonywalny kod jest zazwyczaj tworzony w późniejszych fazach, więc zazwyczaj testowanie dynamiczne może być przeprowadzane dopiero na późniejszych etapach cyklu wytwarzania oprogramowania.

W niektórych iteracyjnych modelach wytwarzania oprogramowania i przyrostowych modelach wytwarzania oprogramowania zakłada się, że każda iteracja dostarcza działający prototyp lub przyrost produktu. Oznacza to, że w każdej iteracji można przeprowadzać zarówno testowanie statyczne, jak i testowanie dynamiczne na wszystkich poziomach testów. Częste dostarczanie przyrostów wymaga szybkiej informacji zwrotnej i szeroko zakrojonego testowania regresji.

Zwinne wytwarzanie oprogramowania zakłada, że w trakcie projektu mogą wystąpić zmiany. Dlatego w projektach zwinnych preferowana jest lekka dokumentacja produktu oraz szeroko zakrojona automatyzacja testów, ułatwiająca testowanie regresji. Ponadto większość testów manualnych jest zazwyczaj wykonywana przy użyciu technik testowania opartych na doświadczeniu (patrz podrozdział 4.4), które nie wymagają wcześniejszego podjęcia szeroko zakrojonej analizy i projektowania testów.

© International Software Testing Qualifications Board © Polish Quality Board

2.1.2. Model cyklu wytwarzania oprogramowania i dobre praktyki testowania

Poniżej przedstawiono dobre praktyki testowania, niezależnie od wybranego modelu cyklu wytwarzania oprogramowania:

- Każdej czynności związanej z wytwarzaniem oprogramowania odpowiada czynność testowa, dzięki czemu wszystkie czynności wytwórcze podlegają kontroli jakości.
- Różne poziomy testów (patrz sekcja 2.2.1) mają określone i różne cele testowania, co pozwala na szeroki zakres testów przy jednoczesnym uniknięciu redundancji.
- Analiza i projektowanie testów dla danego poziomu testów rozpoczynają się podczas odpowiadającej mu fazy cyklu wytwarzania oprogramowania, co zapewnia zgodność z zasadą wczesnego testowania (patrz podrozdział 1.3).
- Testerzy są zaangażowani w przeglądanie produktów pracy natychmiast po udostępnieniu ich projektów, dzięki czemu wcześniejsze testowanie i wykrywanie defektów wspiera przesunięcie w lewo (ang. *shift-left*, patrz sekcja 2.1.5).

2.1.3. Testowanie jako czynnik określający sposób wytwarzania oprogramowania

Wytwarzanie sterowane testami (TDD), wytwarzanie sterowane testami akceptacyjnymi (ATDD) oraz wytwarzanie sterowane zachowaniem (BDD) to podobne podejścia do wytwarzania oprogramowania, w których testy są czynnikiem określającym kierunek prac programistycznych. Każde z tych podejść realizuje zasadę wczesnego testowania (patrz podrozdział 1.3) i stosuje zasadę przesunięcia w lewo (patrz sekcja 2.1.5), ponieważ testy są definiowane przed napisaniem kodu. Podejścia te wspierają iteracyjny model wytwarzania oprogramowania i charakteryzują się następującymi cechami:

Wytwarzanie sterowane testami (TDD):

- Kodowanie kierowane jest poprzez przypadki testowe zamiast przez rozbudowany projekt oprogramowania (Beck 2003)
- Najpierw pisane są testy, następnie kod, tak, aby zaliczyć testy, a na końcu testy i kod są refaktoryzowane.

Wytwarzanie sterowane testami akceptacyjnymi (ATDD) (patrz sekcja 4.5.3):

- Testy wyprowadzane są z kryteriów akceptacji w ramach procesu projektowania systemu (Gärtner 2011).
- Testy są pisane przed wytworzeniem części aplikacji, która ma je pomyślnie przejść.

Wytwarzanie sterowane zachowaniem (BDD):

- Pożądane zachowanie aplikacji wyraża się za pomocą przypadków testowych napisanych w prostej formie, językiem naturalnym, który jest łatwy do zrozumienia dla interesariuszy – zazwyczaj przy użyciu formatu Given/When/Then (Chelimsky 2010).
- Przypadki testowe powinny być automatycznie przekształcane w wykonywalne testy.

W przypadku wszystkich powyższych podejść testy mogą pozostać testami automatycznymi, aby zapewnić jakość kodu w przyszłych adaptacjach bądź refaktoryzacjach.

© International Software Testing Qualifications Board © Polish Quality Board

2.1.4. Metodyka DevOps a testowanie

DevOps to podejście organizacyjne mające na celu stworzenie synergii poprzez współpracę zespołów: wytwarzającego (w tym testowania) i operacyjnego (zajmującego się wdrażaniem i utrzymaniem), by osiągnąć wspólne cele. DevOps wymaga zmiany kulturowej w organizacji, aby wypełnić lukę między tymi dwoma zespołami, traktując ich funkcje jako równo istotne. DevOps promuje autonomię zespołów, szybką informację zwrotną, zintegrowane łańcuchy narzędzi oraz praktyki techniczne, takie jak ciągła integracja (ang. *Continuous Integration*, CI) i ciągłe dostarczanie (ang. *Continuous Delivery*, CD). Umożliwia to zespołom szybsze tworzenie, testowanie i wydawanie wysokiej jakości kodu poprzez dostarczania DevOps (Kim 2016).

Z punktu widzenia testowania korzyści płynące z DevOps to:

- szybka informacja zwrotna na temat jakości kodu oraz tego, czy zmiany mają negatywny wpływ na istniejący kod,
- CI, która promuje przesunięcie w lewo (patrz sekcja 2.1.5), zachęcając programistów do przesyłania wysokiej jakości kodu wraz z testami modułowymi i analizą statyczną,
- promowanie zautomatyzowanych procesów, takich jak CI/CD, które ułatwiają tworzenie stabilnych środowisk testowych,
- zwiększenie widoczności niefunkcjonalnych charakterystyk jakościowych (np. wydajności, niezawodności),
- zmniejszona potrzeba powtarzalnych testów manualnych dzięki automatyzacji poprzez potok dostarczania (ang. *delivery pipeline*),
- minimalizacja ryzyka regresji dzięki skali i zakresowi zautomatyzowanych testów regresji.

DevOps nie jest jednak pozbawiony ryzyk i wyzwań, do których można zaliczyć:

- konieczność zdefiniowania i ustanowienia potoku dostarczania DevOps,
- konieczność wprowadzenia i utrzymywania narzędzi CI / CD,
- konieczność przeznaczenia dodatkowych zasobów na automatyzację testów oraz trudności związane z wdrożeniem i utrzymaniem automatyzacji.

Chociaż DevOps zapewnia wysoki udział testów automatycznych, nadal konieczne są testy manualne, zwłaszcza z perspektywy użytkownika.

2.1.5. Przesunięcie w lewo

Zasada wczesnego testowania (patrz podrozdział 1.3) jest czasami nazywana „przesunięciem w lewo” (ang. *shift left*), ponieważ jest to podejście, w którym testowanie odbywa się na wcześniejszym etapie cyklu wytwarzania oprogramowania. Przesunięcie w lewo sugeruje, że testowanie powinno odbywać się wcześniej (np. nie czekając na implementację kodu lub integrację modułów), ale nie oznacza to, że należy zaniedbywać testowanie na późniejszych etapach cyklu wytwarzania oprogramowania.

Dobre praktyki, które ilustrują, jak osiągnąć „przesunięcie w lewo” w testowaniu, to m.in.:

- przegląd specyfikacji z perspektywy testerów, co pozwala wykryć potencjalne defekty, takie jak niejasności, braki i niespójności,
- pisanie przypadków testowych przed rozpoczęciem pisania kodu i uruchamianie kodu w jarzmie testowym podczas implementacji,

© International Software Testing Qualifications Board © Polish Quality Board

- wykorzystanie CI, a jeszcze lepiej CD, ponieważ zapewnia ono szybką informację zwrotną i automatyczne testy modułowe kodu przesyłanego do repozytorium kodu,
- przeprowadzenie analizy statycznej kodu źródłowego przed rozpoczęciem testowania dynamicznego lub w ramach zautomatyzowanego procesu,
- przeprowadzanie testowania niefunkcjonalnego, zaczynając już od poziomu testów modułowych, jeśli to możliwe (jest to przesunięcie w lewo, ponieważ tego typu testy są zazwyczaj przeprowadzane w późniejszych fazach cyklu wytwarzania oprogramowania, kiedy dostępny jest kompletny system i reprezentatywne środowisko testowe).

Przesunięcie w lewo może wymagać dodatkowych szkoleń, wysiłku lub kosztów na wcześniejszym etapie procesu, ale oczekuje się, że pozwoli zaoszczędzić wysiłek lub koszty na etapach późniejszych.

W przypadku przesunięcia w lewo ważne jest, aby interesariusze byli przekonani do tej koncepcji i ją akceptowali.

2.1.6. Retrospektywy i doskonalenie procesów

Retrospektywy często odbywają się pod koniec projektu lub iteracji, w momencie osiągnięcia kamienia milowego związanego z wydaniem, ale mogą być też organizowane w razie potrzeby. Termin i przebieg retrospektyw zależą od przyjętego modelu cyklu wytwarzania oprogramowania. Podczas retrospektyw uczestnicy (nie tylko testerzy, ale także np. programiści, architekci, właściciele produktu, analitycy biznesowi) omawiają:

- Co się udało i powinno zostać zachowane?
- Co się nie udało i można to poprawić?
- Jak wdrożyć udoskonalenia i wykorzystać pomyślnie zrealizowane elementy w przyszłości?

Wyniki retrospektywy powinny być dokumentowane i zazwyczaj stanowią część sumarycznego raportu z testów (patrz sekcja 5.3.2). Retrospektywy mają kluczowe znaczenie dla pomyślnego wdrożenia ciągłego doskonalenia i ważne jest, aby wszelkie zalecane udoskonalenia były realizowane.

Typowe korzyści płynące z retrospektyw dla testowania obejmują:

- zwiększoną skuteczność i wydajność testów (np. poprzez wdrożenie sugestii dotyczących doskonalenia procesów),
- zwiększoną jakość testaliów (np. poprzez wspólną weryfikację procesów testowych),
- budowanie więzi w zespole i uczenie się (np. dzięki możliwości zgłaszania problemów i proponowania usprawnień),
- poprawę jakości podstawy testów (np. dzięki możliwości identyfikacji i rozwiązania niedociągnięć związanych z jakością i z zakresem wymagań),
- polepszenie współpracy między zespołem wytwórczym a zespołem testowym (np. dzięki regularnemu przeglądowi i optymalizacji zasad współpracy).

2.2. Poziomy testów i typy testów

Poziomy testów to grupy czynności testowych, które są organizowane i zarządzane wspólnie. Każdy poziom testów jest instancją procesu testowego, wykonywaną w odniesieniu

© International Software Testing Qualifications Board © Polish Quality Board

do oprogramowania na danym etapie wytwarzania, od poszczególnych modułów po kompletne systemy lub systemy systemów.

Poziomy testów są powiązane z innymi czynnościami w ramach cyklu wytwarzania oprogramowania. W sekwencyjnych modelach wytwarzania oprogramowania poziomy testów są często definiowane w taki sposób, że kryteria wyjścia jednego poziomu stanowią część kryteriów wejścia do następnego poziomu. W niektórych iteracyjnych modelach cyklu wytwarzania oprogramowania może to nie mieć zastosowania. Czynności związane z wytwarzaniem oprogramowania mogą obejmować wiele poziomów testów, a poziomy testów mogą na siebie nachodzić w czasie.

Typy testów to grupy czynności testowych związanych z określonymi charakterystykami jakościowymi, a większość tych czynności można wykonać na każdym poziomie testów.

2.2.1. Poziomy testów

W niniejszym sylabusie opisano pięć następujących poziomów testów:

- **Testowanie modułowe** (zwane również testowaniem jednostkowym) koncentruje się na testowaniu modułów w izolacji. Często wymaga to specjalnego wsparcia, takiego jak jarzma testowe lub struktury do testów jednostkowych. Testowanie modułowe jest zazwyczaj wykonywane przez programistów w ich środowiskach programistycznych.
- **Testowanie integracji modułów** (zwane również testowaniem integracji jednostek) koncentruje się na testowaniu interfejsów i interakcji między modułami. Testowanie to jest w dużym stopniu uzależnione od strategii integracji, takiej jak strategia wstępująca (ang. *bottom-up*), zstępująca (ang. *top-down*) lub wielkiego wybuchu (ang. *big-bang*).
- **Testowanie systemowe** koncentruje się na ogólnym zachowaniu i możliwościach systemu lub produktu jako całości, często obejmując testowanie funkcjonalne zadań typu *end-to-end* oraz testowanie niefunkcjonalne charakterystyk jakościowych. W przypadku niektórych niefunkcjonalnych charakterystyk jakościowych, zaleca się testowanie ich w reprezentatywnym środowisku testowym (np. użyteczność). Możliwe jest również wykorzystanie symulacji podsystemów. Testowanie systemu może być przeprowadzane przez niezależny zespół testowy i jest związane ze specyfikacjami systemu.
- **Testowanie integracji systemów** koncentruje się na testowaniu interfejsów między testowanym systemem a innymi systemami i usługami zewnętrznymi. Testy integracji systemów wymagają odpowiednich środowisk testowych, najlepiej podobnych do środowiska operacyjnego.
- **Testowanie akceptacyjne** koncentruje się na walidacji i wykazaniu gotowości do wdrożenia, co oznacza, że system spełnia potrzeby biznesowe użytkownika. Idealnie, jeśli testy akceptacyjne są przeprowadzane przez docelowych użytkowników. Główne formy testów akceptacyjnych to: testowanie akceptacyjne użytkownika (ang. *User Acceptance Testing*, UAT), operacyjne testy akceptacyjne (ang. *Operational Acceptance Testing*, OAT), testowanie akceptacyjne zgodności z umową, testowanie akceptacyjne zgodności z prawem, testowanie alfa i testowanie beta.

© International Software Testing Qualifications Board © Polish Quality Board

Poziomy testów rozróżnia się na podstawie poniższej, niewyczerpującej listy atrybutów, aby uniknąć nakładania się działań testowych:

- przedmiot testów,
- cele testów,
- podstawa testów,
- defekty i awarie,
- podejście i odpowiedzialności.

2.2.2. Typy testów

Istnieje wiele typów testów. W niniejszym sylabusie omówiono cztery z nich.

Testowanie funkcjonalne ocenia funkcje, które powinien wykonywać moduł lub system. Funkcje te opisują, „co" powinien robić przedmiot testów. Głównym celem testowania funkcjonalnego jest sprawdzenie kompletności funkcjonalnej, poprawności funkcjonalnej i adekwatności funkcjonalnej.

Testowanie niefunkcjonalne ocenia inne niż funkcjonalność charakterystyki modułu lub systemu. Testowanie niefunkcjonalne sprawdza, „jak dobrze" zachowuje się system. Głównym celem testów niefunkcjonalnych jest sprawdzenie niefunkcjonalnych charakterystyk jakościowych. Norma ISO/IEC 25010 zawiera następującą klasyfikację niefunkcjonalnych charakterystyk jakościowych:

- wydajność,
- kompatybilność,
- użyteczność (znana również jako zdolność do interakcji),
- niezawodność,
- zabezpieczenia,
- utrzymywalność,
- przenaszalność (znana również jako elastyczność),
- bezpieczeństwo.

Czasami wskazane jest rozpoczęcie testów niefunkcjonalnych na wczesnym etapie cyklu wytwarzania oprogramowania (np. w ramach przeglądów lub testowania modułowego). Podstawą wielu testów niefunkcjonalnych są testy funkcjonalne – w praktyce wykorzystywane są te same testy funkcjonalne, ale celem jest sprawdzenie, czy podczas wykonywania danej funkcji spełniane są wymagania niefunkcjonalne (np. sprawdzenie, czy funkcja jest wykonywana w określonym czasie lub czy można ją przenieść na nową platformę). Późne wykrycie defektów niefunkcjonalnych może stanowić poważne zagrożenie dla powodzenia projektu. Testy niefunkcjonalne wymagają czasami bardzo specyficznego środowiska testowego, takiego jak laboratorium użyteczności do testowania użyteczności.

Testowanie czarnoskrzynkowe (patrz podrozdział 4.2) jest oparte na specyfikacjach. Testy wyprowadza się na podstawie dokumentacji niezwiązanej z wewnętrzną strukturą przedmiotu testów. Głównym celem testowania czarnoskrzynkowego jest sprawdzenie zachowania systemu pod kątem zgodności z jego specyfikacją.

Testowanie białoskrzynkowe (patrz podrozdział 4.3) opiera się na strukturze przedmiotu testów. Testy wyprowadza się z implementacji systemu lub jego wewnętrznej struktury (np. kodu,

© International Software Testing Qualifications Board © Polish Quality Board

architektury, przepływów pracy, przepływów danych). Głównym celem testowania białoskrzynkowego jest uzyskanie akceptowalnego poziomu pokrycia testami tej struktury.

Wszystkie cztery wyżej wymienione typy testów mogą być stosowane na wszystkich poziomach testów, chociaż działania na każdym poziomie będą inaczej ukierunkowane. Do wyprowadzenia warunków testowych i przypadków testowych dla wszystkich wymienionych typów testów można stosować różne techniki testowania.

2.2.3. Testowanie potwierdzające i testowanie regresji

W modułach lub systemach często wprowadza się zmiany w celu ich ulepszenia poprzez dodanie nowej funkcjonalności lub naprawienie poprzez usunięcie defektu. Testowanie powinno wówczas obejmować testowanie potwierdzające i testowanie regresji.

Testowanie potwierdzające ma na celu sprawdzenie, czy wykryty uprzednio defekt został pomyślnie usunięty. W zależności od ryzyka można przetestować naprawioną wersję oprogramowania na kilka sposobów, w tym poprzez:

- wykonanie wszystkich testów, które wcześniej zakończyły się niezaliczeniem z powodu defektu,
- dodanie nowych testów obejmujących wszelkie zmiany, które były konieczne do usunięcia defektu.

Jeśli jednak na usunięcie defektu brakuje czasu lub pieniędzy, testowanie potwierdzające może ograniczać się do wykonania kroków testowych, które powinny odtworzyć awarię spowodowaną defektem, i sprawdzenia, czy awaria już nie występuje.

Testowanie regresji ma na celu sprawdzenie, że zmiana (w tym poprawka, która była już przedmiotem testowania potwierdzającego) nie spowodowała żadnych niekorzystnych skutków. Te niekorzystne skutki mogą mieć wpływ na ten sam moduł, w którym wprowadzono zmianę, inne moduły w tym samym systemie, a nawet inne systemy. Testowanie regresji nie musi zatem ograniczać się do samego przedmiotu testów, może również odnosić się do środowiska, w którym przedmiot testów się znajduje. Dlatego zaleca się najpierw przeprowadzenie analizy wpływu szacującej zasięg testowania regresji, aby rozpoznać, które części oprogramowania mogą zostać dotknięte zmianami.

Zestawy testów regresji są uruchamiane wiele razy i zazwyczaj liczba związanych z nimi przypadków testowych wzrasta wraz z każdą iteracją lub wydaniem. Dlatego testowanie regresji jest naturalnym kandydatem do automatyzacji. Automatyzację testów należy rozpocząć na wczesnym etapie projektu. Automatyzacja testów regresji jest również dobrą praktyką w przypadku stosowania CI, np. w DevOps (patrz sekcja 2.1.4). W zależności od sytuacji może ona obejmować testy regresji na różnych poziomach testów.

Testowanie potwierdzające oraz testowanie regresji dla przedmiotu testów jest wykonywane na wszystkich poziomach testów, na których usuwano defekty lub wprowadzano zmiany.

2.3. Testowanie pielęgnacyjne

Istnieją różne kategorie pielęgnacji. Może ona mieć charakter korygujący (działania naprawcze), adaptacyjny (dostosowanie do zmian w środowisku) lub doskonalący (np. poprawiający wydajność lub łatwość utrzymania, patrz norma ISO/IEC 14764). Pielęgnacja może zatem

© International Software Testing Qualifications Board © Polish Quality Board

obejmować zarówno planowane wydania/wdrożenia, jak i nieplanowane doraźne poprawki (ang. *hot fix*). Przed dokonaniem zmiany można przeprowadzić analizę wpływu, aby ustalić, czy zmiana powinna zostać wprowadzona, w oparciu o potencjalne konsekwencje w innych obszarach systemu. Testowanie zmian w już działającym systemie obejmuje zarówno ocenę powodzenia wdrożenia zmiany, jak i sprawdzenie ewentualnych regresji w częściach systemu, które pozostały niezmienione (co zazwyczaj dotyczy większości systemu).

Zakres testowania pielęgnacyjnego zależy zazwyczaj od:

- poziomu ryzyka związanego ze zmianą,
- wielkości istniejącego systemu,
- wielkości zmiany.

Czynniki wywołujące pielęgnację i testowanie pielęgnacyjne można sklasyfikować w następujący sposób:

- Modyfikacje, takie jak planowane ulepszenia (tj. nowe wersje systemu), zmiany korygujące lub doraźne poprawki.
- Aktualizacje lub migracje środowiska operacyjnego, np. z jednej platformy na drugą, które mogą wymagać testów związanych z nowym środowiskiem, a także ze zmienionym oprogramowaniem lub testów konwersji danych, gdy dane z innej aplikacji są migrowane do pielęgnowanego systemu.
- Wycofanie z eksploatacji, np. gdy aplikacja osiąga koniec swojego cyklu życia. Wycofanie systemu z eksploatacji może wymagać testowania archiwizacji danych, jeśli wymagane są długie okresy przechowywania danych. Testowanie procedur przywracania i odzyskiwania danych po archiwizacji może być również konieczne w przypadku, gdy w okresie archiwizacji wymagany jest dostęp do określonych danych.

© International Software Testing Qualifications Board © Polish Quality Board

3. Testowanie statyczne – 80 minut

Słowa kluczowe

analiza statyczna, anomalia, inspekcja, przegląd, przegląd formalny, przegląd nieformalny, przegląd techniczny, przejrzenie, testowanie dynamiczne, testowanie statyczne

Cele nauczania dla rozdziału 3

3.1 Podstawy testowania statycznego

FL-3.1.1 (K1) Rozpoznawać typy produktów pracy, które mogą być badane za pomocą testowania statycznego.

FL-3.1.2 (K2) Wyjaśnić korzyści wynikające z testowania statycznego.

FL-3.1.3 (K2) Porównać i zestawiać ze sobą testowanie statyczne i testowanie dynamiczne.

3.2 Informacje zwrotne i proces przeglądu

FL-3.2.1 (K1) Pamiętać korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy.

FL-3.2.2 (K2) Podsumować czynności wykonywane w ramach procesu przeglądu.

FL-3.2.3 (K1) Pamiętać, jakie obowiązki są przypisane do najważniejszych ról w trakcie wykonywania przeglądów.

FL-3.2.4 (K2) Porównać i zestawiać ze sobą różne typy przeglądów.

FL-3.2.5 (K1) Pamiętać, jakie czynniki decydują o powodzeniu przeglądu.

© International Software Testing Qualifications Board © Polish Quality Board

3.1. Podstawy testowania statycznego

W przeciwieństwie do testowania dynamicznego, w testowaniu statycznym nie uruchamia się testowanego oprogramowania. Kod, specyfikacja, projekt architektury lub inne produkty pracy są oceniane poprzez badanie manualne (przeglądy) lub przy pomocy narzędzia (analiza statyczna). Cele testowania statycznego obejmują poprawę jakości, wykrywanie defektów oraz ocenę takich charakterystyk jak czytelność, kompletność, poprawność, testowalność i spójność. Testowanie statyczne można stosować zarówno do weryfikacji, jak i walidacji.

Testerzy, przedstawiciele jednostek biznesowych (właściciel produktu, analityk biznesowy itp.) i programiści współpracują podczas sesji mapowania przykładów (ang. *example mapping*), wspólnego pisania historyjek użytkownika i sesji doskonalenia backlogu (ang. *backlog refinement sessions*), aby zapewnić, że historyjki użytkownika i powiązane z nimi produkty pracy spełniają określone kryteria, np. definicję gotowości (ang. *Definition of Ready*, DoR, patrz sekcja 5.1.3). Techniki przeglądu mogą być stosowane w celu zapewnienia, że historyjki użytkownika są kompletne i zrozumiałe oraz zawierają testowalne kryteria akceptacji. Zadając właściwe pytania, testerzy badają, kwestionują i doskonalą proponowane historyjki użytkownika.

Analiza statyczna pozwala zidentyfikować problemy przed testowaniem dynamicznym, często wymagając mniejszego wysiłku, ponieważ nie są potrzebne żadne przypadki testowe i zazwyczaj wykorzystuje się narzędzia (patrz rozdział 6). Analiza statyczna stanowi często część procesu ciągłej integracji (patrz sekcja 2.1.4). Chociaż jest ona w dużej mierze wykorzystywana do wykrywania konkretnych defektów w kodzie, służy również do oceny łatwości pielęgnacji i poziomu zabezpieczeń. Inne przykłady narzędzi do analizy statycznej to narzędzia do sprawdzania pisowni i do oceny czytelności tekstu.

3.1.1. Produkty pracy badane metodą testowania statycznego

Za pomocą testowania statycznego można zbadać prawie każdy produkt pracy. Przykłady obejmują dokumenty specyfikacji wymagań, kod źródłowy, plany testów, przypadki testowe, elementy backlogu produktu, karty opisu testów, dokumentację projektu, umowy i modele.

Każdy produkt pracy, który można przeczytać i zrozumieć, może być przedmiotem przeglądu. Jednak w przypadku analizy statycznej produkty pracy muszą mieć strukturę, względem której można je sprawdzić (np. modele, kod lub tekst o formalnej składni).

Produkty pracy, które nie nadają się do testowania statycznego, to między innymi te, które są trudne do interpretacji przez ludzi i nie powinny być analizowane za pomocą narzędzi (np. kod wykonywalny lub kod innych firm, którego nie można badać z powodów prawnych).

3.1.2. Korzyści wynikające z testowania statycznego

Testowanie statyczne pozwala wykryć defekty na najwcześniejszych etapach cyklu wytwarzania oprogramowania, spełniając zasadę wczesnego testowania (patrz podrozdział 1.3). Pozwala ono również identyfikować defekty, których nie można wykryć za pomocą testowania dynamicznego (np. martwy kod, błędnie zastosowane wzorce projektowe, defekty w niewykonywalnych produktach pracy).

Testowanie statyczne umożliwia ocenę jakości produktów pracy i budowanie zaufania do nich. Dzięki weryfikacji udokumentowanych wymagań interesariusze mogą również upewnić się,

© International Software Testing Qualifications Board © Polish Quality Board

że wymagania te opisują ich rzeczywiste potrzeby. Ponieważ testowanie statyczne można przeprowadzać na wczesnym etapie cyklu wytwarzania oprogramowania, zaangażowani w ten proces interesariusze mogą wypracować wspólny punkt widzenia. Inną ważną korzyścią jest usprawnienie wymiany informacji pomiędzy interesariuszami. Z tego powodu zaleca się, aby w proces testowania statycznego zaangażowane było możliwie szerokie ich grono.

Mimo że wdrożenie przeglądów może być kosztowne, całkowite koszty projektu są zazwyczaj znacznie niższe niż w przypadku braku przeglądów, ponieważ trzeba poświęcić mniej czasu i wysiłku na usuwanie defektów w późniejszych fazach projektu.

Analiza statyczna pozwala wykrywać niektóre defekty w kodzie skuteczniej niż testowanie dynamiczne, co zazwyczaj skutkuje mniejszą liczbą tego rodzaju defektów i mniejszym ogólnym wysiłkiem związanym z wytwarzaniem oprogramowania.

3.1.3. Różnice między testowaniem statycznym a dynamicznym

Testowanie statyczne i testowanie dynamiczne wzajemnie się uzupełniają. Mają podobne cele, takie jak wykrywanie defektów w produktach pracy (patrz sekcja 1.1.1), ale różnią się pod pewnymi względami opisanymi poniżej:

- Zarówno testowanie statyczne, jak i testowanie dynamiczne (z analizą awarii), prowadzi do wykrycia defektów, jednak istnieją pewne rodzaje defektów wykrywalne tylko za pomocą testowania statycznego lub tylko za pomocą testowania dynamicznego.
- Testowanie statyczne wykrywa defekty bezpośrednio, natomiast testowanie dynamiczne powoduje występowanie awarii, których późniejsza analiza pozwala identyfikować związane z nimi defekty.
- Testowanie statyczne może łatwiej wykrywać defekty znajdujące się na ścieżkach kodu, rzadko wykonywanych lub trudnych do osiągnięcia w testowaniu dynamicznym.
- Testowanie statyczne można stosować do niewykonywalnych produktów pracy, natomiast testowanie dynamiczne – tylko do wykonywalnych produktów pracy.
- Testowanie statyczne może być stosowane do pomiaru charakterystyk jakościowych niezależnych od wykonywania kodu (np. utrzymywalność), natomiast testowanie dynamiczne może być stosowane do pomiaru charakterystyk jakościowych, które są zależne od wykonywania kodu (np. wydajność).

Typowe defekty, które można wykryć łatwiej lub taniej za pomocą testowania statycznego, obejmują:

- defekty w wymaganiach (np. niespójności, niejasności, sprzeczności, pominięcia, niedokładności, powtórzenia),
- defekty w projekcie (np. nieefektywne struktury baz danych, słaba modularyzacja),
- niektóre typy defektów w kodzie (np. zmienne o nieokreślonych wartościach, zmienne niezadeklarowane, nieosiągalny lub zduplikowany kod, nadmierna złożoność kodu),
- odstępstwa od norm (np. brak zgodności z konwencjami nazewnictwa w standardach kodowania),
- nieprawidłowe specyfikacje interfejsów (np. zła liczba, typ lub kolejność parametrów),
- określone rodzaje luk w zabezpieczeniach (np. przepełnienie bufora),
- braki lub nieścisłości w pokryciu podstawy testów (np. brakujące testy dla kryterium akceptacji).

© International Software Testing Qualifications Board © Polish Quality Board

3.2. Informacje zwrotne i proces przeglądu

3.2.1. Korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy

Wczesne i częste informacje zwrotne pozwalają na wczesne zgłaszanie potencjalnych problemów związanych z jakością. Jeśli zaangażowanie interesariuszy w cyklu wytwarzania oprogramowania jest niewielkie, opracowywany produkt może nie spełniać pierwotnej lub aktualnej wizji interesariuszy. Niezrealizowanie oczekiwań interesariuszy może skutkować kosztownymi przeróbkami, niedotrzymaniem terminów, wzajemnym obwinianiem się, a nawet całkowitym niepowodzeniem projektu.

Częste informacje zwrotne od interesariuszy we wszystkich fazach cyklu wytwarzania oprogramowania mogą zapobiec nieporozumieniom dotyczącym wymagań i zapewnić, że zmiany w wymaganiach zostaną zrozumiane i wdrożone wcześniej. Pomaga to zespołowi wytwórczemu lepiej zrozumieć, co tworzy. Pozwala mu także skupić się na tych funkcjach, które zapewniają największą wartość dla interesariuszy i mają najbardziej pozytywny wpływ na zidentyfikowane ryzyka.

3.2.2. Czynności wykonywane w procesie przeglądu

Norma ISO/IEC 20246 definiuje ogólny proces przeglądu, który zapewnia ustrukturyzowane, ale elastyczne ramy, na podstawie których można dostosować konkretny proces przeglądu do danej sytuacji. Jeśli wymagany przegląd ma bardziej formalny charakter, konieczne będzie wykonanie większej liczby zadań opisanych poniżej dla różnych działań.

Wiele produktów pracy ma zbyt duży rozmiar, aby można je było objąć jednym przeglądem. Z tego powodu proces przeglądu może być przeprowadzany wielokrotnie, aby przeanalizować całość produktu pracy.

W procesie przeglądu można wyróżnić następujące czynności:

- **Planowanie**. Podczas fazy planowania należy określić zakres przeglądu, który obejmuje cel, produkt pracy podlegający przeglądowi, charakterystyki jakościowe podlegające ocenie, obszary, na których należy się skupić, kryteria wyjścia, informacje pomocnicze (np. wykorzystywane normy), wysiłek oraz ramy czasowe przeglądu.
- **Rozpoczęcie przeglądu** polega na upewnieniu się, że wszystkie zaangażowane osoby i niezbędne elementy są gotowe do rozpoczęcia przeglądu. Obejmuje to sprawdzenie, czy każdy uczestnik ma dostęp do przeglądanego produktu pracy, rozumie swoją rolę i obowiązki oraz otrzymał wszystkie informacje niezbędne do przeprowadzenia przeglądu.
- **Przegląd indywidualny**. Każdy przeglądający dokonuje indywidualnego przeglądu w celu oceny jakości przeglądanego produktu pracy oraz zidentyfikowania anomalii, zaleceń i pytań, stosując jedną lub więcej technik przeglądu (np. przegląd oparty na liście kontrolnej, przegląd oparty na scenariuszach). Norma ISO/IEC 20246 zawiera bardziej szczegółowe informacje na temat różnych technik przeglądu. Przeglądający rejestrują wszystkie zidentyfikowane przez siebie anomalie, zalecenia i pytania.
- **Przekazanie informacji i analiza**. Ponieważ zidentyfikowane podczas przeglądu anomalie niekoniecznie muszą być defektami, wszystkie anomalie należy przeanalizować i omówić, określić ich status, wskazać wymagane działania względem anomalii oraz

© International Software Testing Qualifications Board © Polish Quality Board

wyznaczyć osobę za to odpowiedzialną. Zazwyczaj czynności te odbywają się podczas spotkania przeglądowego, podczas którego uczestnicy decydują również o poziomie jakości przeglądanego produktu pracy oraz dalszych wymaganych działaniach. W szczególności działania te mogą obejmować przeprowadzenie kolejnego przeglądu.
- **Usunięcie defektów i raportowanie**. Dla każdego zidentyfikowanego defektu należy sporządzić raport o defekcie, aby umożliwić podjęcie działań naprawczych. Po osiągnięciu kryteriów wyjścia produkt pracy można uznać za zaakceptowany. Wyniki przeglądu są raportowane.

3.2.3. Role i obowiązki w przeglądach

W przeglądach biorą udział różni interesariusze, którzy mogą pełnić różne role. Główne role i związane z nimi obowiązki są następujące:

- **Kierownik** – decyduje, co ma zostać poddane przeglądowi, i zapewnia zasoby takie jak personel i czas przeznaczony na przegląd.
- **Autor** – tworzy oraz usuwa defekty z produktu pracy poddawanego przeglądowi.
- **Moderator** (zwany również facylitatorem) – zapewnia skuteczne prowadzenie spotkań przeglądowych, w tym mediację, zarządzanie czasem i bezpieczne środowisko, w którym każdy może swobodnie zabierać głos.
- **Protokolant** (zwany również rejestrującym) – dokumentuje uwagi przeglądających i rejestruje informacje dotyczące przeglądu, takie jak decyzje i nowe nieprawidłowości wykryte podczas spotkania przeglądowego.
- **Przeglądający** – przeprowadza przegląd. Może to być osoba pracująca nad projektem, ekspert merytoryczny lub dowolny inny interesariusz.
- **Lider przeglądu** – ponosi ogólną odpowiedzialność za przegląd, np. decyduje, kto będzie w nim uczestniczył, oraz organizuje miejsce przeglądu i wyznacza jego termin.

Norma ISO/IEC 20246 opisuje również inne, bardziej szczegółowe role.

3.2.4. Typy przeglądów

Istnieje wiele typów przeglądów, od nieformalnych po formalne. Wymagany poziom sformalizowania zależy od takich czynników jak stosowany cykl wytwarzania oprogramowania, dojrzałość procesu wytwórczego, krytyczność i złożoność przeglądanego produktu pracy, wymogi prawne lub regulacyjne oraz potrzeba prowadzenia ścieżki audytu. Ten sam produkt pracy może być poddawany różnym typom przeglądów, np. najpierw nieformalnym, a później bardziej sformalizowanym.

Wybór odpowiedniego typu przeglądu ma kluczowe znaczenie dla osiągnięcia wymaganych celów przeglądu (patrz sekcja 3.2.5). Wybór ten opiera się nie tylko na celach, ale także na takich czynnikach jak potrzeby projektu, dostępne zasoby, rodzaj produktu pracy, zidentyfikowane ryzyka, dziedzina biznesowa oraz kultura organizacyjna firmy.

Niniejszy sylabus omawia następujące cztery typy przeglądów:

- **Przegląd nieformalny** (ang. *informal review*). Nie przebiega według określonego procesu i nie wymaga formalnego dokumentowania wyników. Jego głównym celem jest wykrywanie anomalii.

© International Software Testing Qualifications Board © Polish Quality Board

- **Przejrzenie** (ang. *walkthrough*). Jest prowadzone przez autora. Może służyć wielu celom takim jak ocena jakości i budowanie zaufania do produktu pracy, edukowanie przeglądających, osiągnięcie konsensusu, generowanie nowych pomysłów, motywowanie autorów do rozwoju i stworzenie im warunków do tego oraz wykrywanie anomalii. Przed rozpoczęciem przejrzenia przeglądający mogą przeprowadzić przegląd indywidualny, ale nie jest to konieczne.
- **Przegląd techniczny** (ang. *technical review*). Prowadzony przez moderatora i wykonywany przez przeglądających, którzy dysponują odpowiednimi kwalifikacjami technicznymi. Celem przeglądu technicznego jest osiągnięcie konsensusu i podjęcie decyzji dotyczących problemu technicznego, ale także wykrywanie anomalii, ocena jakości i budowanie zaufania do produktu pracy, generowanie nowych pomysłów oraz motywowanie autorów do rozwoju i stworzenie im warunków do tego.
- **Inspekcja** (ang. *inspection*). Najbardziej formalny rodzaj przeglądu, przebiega zgodnie z pełnym procesem opisanym w sekcji 3.2.2. Głównym celem jest wykrycie jak największej liczby anomalii. Inne cele to ocena jakości, budowanie zaufania do produktu pracy oraz motywowanie autorów do rozwoju i stworzenie im warunków do tego. Podczas inspekcji zbiera się metryki wykorzystywane później do doskonalenia cyklu wytwarzania oprogramowania, w tym samego procesu inspekcji. W ramach inspekcji autor nie może pełnić roli lidera przeglądu ani protokolanta.

3.2.5. Czynniki sukcesu związane z przeglądami

Czynniki decydujące o sukcesie przeglądów to między innymi:

- określenie jasnych celów i mierzalnych kryteriów wyjścia (ocena uczestników nigdy nie powinna być celem przeglądu),
- wybór odpowiedniego typu przeglądu, aby umożliwić osiągnięcie założonych celów i dostosować przegląd do rodzaju produktu pracy, uczestników przeglądu, potrzeb projektu i kontekstu,
- przeprowadzanie przeglądów dla małych partii materiału, aby przeglądający nie tracili koncentracji podczas przeglądu indywidualnego lub spotkania przeglądowego (jeśli się odbywa),
- przekazywanie interesariuszom i autorom informacji zwrotnych z przeglądów, aby mogli doskonalić produkt i usprawniać swoje działania (patrz sekcja 3.2.1),
- zapewnienie uczestnikom odpowiedniego czasu na przygotowanie się do przeglądu,
- wsparcie kierownictwa dla procesu przeglądu,
- włączenie przeglądów w kulturę organizacyjną w celu promowania poszerzania wiedzy i doskonalenia procesów,
- zapewnienie odpowiednich szkoleń wszystkim uczestnikom, aby wiedzieli, jak wypełniać swoje role,
- sprawne przeprowadzanie spotkań.

© International Software Testing Qualifications Board © Polish Quality Board

4. Analiza i projektowanie testów (390 min)

Słowa kluczowe

analiza wartości brzegowych, białoskrzynkowa technika testowania, czarnoskrzynkowa technika testowania, element pokrycia, kryteria akceptacji, podejście do testowania oparte na współpracy, podział na klasy równoważności, pokrycie, pokrycie gałęzi, pokrycie instrukcji kodu, technika testowania oparta na doświadczeniu, technika testowania, testowanie eksploracyjne, testowanie przejść pomiędzy stanami, testowanie w oparciu o tablicę decyzyjną, testowanie w oparciu o listę kontrolną, wytwarzanie sterowane testami akceptacyjnymi, zgadywanie błędów

Cele nauczania dla rozdziału 4

4.1 Ogólna charakterystyka technik testowania

FL-4.1.1 (K2) Rozróżniać czarnoskrzynkowe techniki testowania, białoskrzynkowe techniki testowania oraz techniki testowania oparte na doświadczeniu.

4.2 Czarnoskrzynkowe techniki testowania

FL-4.2.1 (K3) Zastosować technikę podziału na klasy równoważności, aby zaprojektować przypadki testowe.

FL-4.2.2 (K3) Zastosować technikę analizy wartości brzegowych, aby zaprojektować przypadki testowe.

FL-4.2.3 (K3) Zastosować technikę testowania w oparciu o tablicę decyzyjną, aby zaprojektować przypadki testowe.

FL-4.2.4 (K3) Zastosować technikę testowania przejść pomiędzy stanami, aby zaprojektować przypadki testowe.

4.3 Białoskrzynkowe techniki testowania

FL-4.3.1 (K2) Wyjaśnić technikę testowania instrukcji.

FL-4.3.2 (K2) Wyjaśnić technikę testowania gałęzi.

FL-4.3.3 (K2) Wyjaśnić korzyści wynikające z testowania białoskrzynkowego.

4.4 Techniki testowania oparte na doświadczeniu

FL-4.4.1 (K2) Wyjaśnić technikę zgadywania błędów.

FL-4.4.2 (K2) Wyjaśnić technikę testowania eksploracyjnego.

FL-4.4.3 (K2) Wyjaśnić technikę testowania w oparciu o listę kontrolną.

4.5 Podejścia do testowania oparte na współpracy

FL-4.5.1 (K2) Wyjaśnić, w jaki sposób należy pisać historyjki użytkownika we współpracy z programistami i przedstawicielami jednostek biznesowych.

FL-4.5.2 (K2) Klasyfikować różne sposoby pisania kryteriów akceptacji.

FL-4.5.3 (K3) Zastosować metodę wytwarzania sterowanego testami akceptacyjnymi (ATDD), aby zaprojektować przypadki testowe.

© International Software Testing Qualifications Board © Polish Quality Board

4.1. Ogólna charakterystyka technik testowania

Techniki testowania wspierają testera w analizie testów (co testować) i projektowaniu testów (jak testować). Pomagają w opracowaniu stosunkowo niewielkiego, ale wystarczającego zestawu przypadków testowych w sposób systematyczny. Pomagają również testerowi w definiowaniu warunków testowych, identyfikowaniu elementów pokrycia i identyfikowaniu danych testowych podczas analizy i projektowania testów. Więcej informacji na temat technik testowania można znaleźć w normie ISO/IEC/IEEE 29119-4 oraz w publikacjach (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

W niniejszym sylabusie techniki testowania są klasyfikowane jako czarnoskrzynkowe, białoskrzynkowe oraz oparte na doświadczeniu.

Czarnoskrzynkowe techniki testowania (zwane również technikami opartymi na specyfikacji) opierają się na analizie zachowania przedmiotu testów bez odniesienia do jego wewnętrznej struktury. Dlatego też przypadki testowe są niezależne od sposobu implementacji oprogramowania. W konsekwencji, jeśli implementacja ulegnie zmianie, ale wymagane zachowanie pozostanie takie samo, przypadki testowe nadal będą aktualne.

Białoskrzynkowe techniki testowania (zwane również technikami opartymi na strukturze) opierają się na analizie wewnętrznej struktury i przetwarzania przedmiotu testów. Ponieważ przypadki testowe są zależne od sposobu zaprojektowania oprogramowania, można je tworzyć dopiero po zaprojektowaniu lub implementacji przedmiotu testów.

Techniki testowania oparte na doświadczeniu wykorzystują wiedzę i doświadczenie testerów do projektowania i implementacji testów. Skuteczność tych technik zależy w dużej mierze od umiejętności testera. Techniki te pozwalają wykrywać defekty, które mogły zostać pominięte przy użyciu czarnoskrzynkowych i białoskrzynkowych technik testowania. Dlatego też techniki testowania oparte na doświadczeniu stanowią uzupełnienie powyższych technik.

4.2. Czarnoskrzynkowe techniki testowania

W poniższych sekcjach omówiono cztery powszechnie stosowane techniki czarnoskrzynkowe:

- podział na klasy równoważności,
- analiza wartości brzegowych,
- testowanie w oparciu o tablicę decyzyjną,
- testowanie przejść pomiędzy stanami.

4.2.1. Podział na klasy równoważności

Technika podziału na klasy równoważności dzieli dane na zbiory zwane klasami równoważności w oparciu o założenie, że wszystkie elementy danej klasy równoważności są przetwarzane w ten sam sposób przez przedmiot testów. Teoria stojąca za tą techniką zakłada więc, że jeśli przypadek testowy, który testuje jedną wartość z klasy równoważności, wykryje defekt, to defekt ten powinien zostać wykryty również przez przypadki testowe, które testują dowolną inną wartość z tej samej klasy równoważności. W związku z tym wystarczy jeden test dla każdej klasy równoważności.

© International Software Testing Qualifications Board © Polish Quality Board

Klasy równoważności można zidentyfikować dla każdego elementu danych związanego z przedmiotem testów, w tym danych wejściowych, wyjściowych, elementów konfiguracji, wartości wewnętrznych, wartości związanych z czasem i parametrów interfejsu. Klasy równoważności mogą być ciągłe lub dyskretne, uporządkowane lub nieuporządkowane, skończone lub nieskończone. Klasy równoważności muszą być rozłączne (nie mogą się nakładać) i niepuste.

W przypadku prostych przedmiotów testów podział na klasy równoważności zazwyczaj nie nastręcza trudności, ale w praktyce zrozumienie, w jaki sposób przedmiot testów będzie traktował różne wartości, jest często skomplikowane. Dlatego przy dokonywaniu podziału na klasy równoważności należy zachować ostrożność.

Klasa równoważności zawierająca wartości poprawne nazywana jest poprawną klasą równoważności, a ta zawierająca wartości niepoprawne – niepoprawną klasą równoważności. Definicje wartości poprawnych i niepoprawnych mogą się różnić w zależności od zespołów i organizacji. Na przykład wartości poprawne mogą być interpretowane jako te, które powinny być przetwarzane przez przedmiot testów, lub jako te, dla których specyfikacja definiuje sposób ich przetwarzania. Wartości niepoprawne mogą być interpretowane jako te, które powinny być ignorowane lub odrzucane przez przedmiot testów, lub jako te, dla których specyfikacja nie definiuje przetwarzania przez przedmiot testów.

W technice podziału na klasy równoważności elementami pokrycia są klasy równoważności. Aby osiągnąć 100% pokrycia przy użyciu tej techniki, przypadki testowe muszą sprawdzić wszystkie zidentyfikowane klasy równoważności (w tym niepoprawne), pokrywając każdą z nich co najmniej raz. Pokrycie mierzy się jako liczbę klas równoważności sprawdzonych przez co najmniej jeden przypadek testowy, podzieloną przez całkowitą liczbę zidentyfikowanych klas równoważności, a uzyskaną wartość wyraża się w procentach.

Często element testowy zawiera wiele grup klas równoważności (np. elementy testowe z więcej niż jednym parametrem wejściowym), co oznacza, że przypadek testowy pokrywa jednocześnie klasy równoważności pochodzące z różnych grup klas równoważności. Najprostszym kryterium pokrycia w takim przypadku jest pokrycie „każdy wybór” (ang. *each choice*) (Ammann 2016). Pokrycie to wymaga, aby przypadki testowe sprawdzały każdą klasę równoważności z każdej grupy klas przynajmniej raz. Pokrycie „każdy wybór” nie bierze pod uwagę kombinacji klas równoważności.

4.2.2. Analiza wartości brzegowych

Analiza wartości brzegowych to technika testowania oparta na sprawdzaniu wartości leżących na brzegach klas równoważności. Dlatego może być stosowana tylko w przypadku klas uporządkowanych. Wartościami brzegowymi klasy równoważności są jej minimalna i maksymalna wartość. W przypadku tej techniki, jeśli dwa elementy należą do tej samej klasy równoważności, wszystkie elementy pomiędzy nimi również muszą należeć do tej klasy.

Analiza wartości brzegowych koncentruje się na wartościach brzegowych, ponieważ programiści często popełniają błędy (tzw. „błędy o jeden”) powodujące defekty związane z tymi właśnie wartościami. Typowe defekty wykrywane przez analizę wartości brzegowych znajdują się zatem w miejscach, gdzie zaimplementowane granice klas równoważności są umieszczone w pozycjach powyżej lub poniżej zamierzonych pozycji lub są całkowicie pominięte.

© International Software Testing Qualifications Board © Polish Quality Board

Niniejszy sylabus omawia dwie wersje analizy wartości brzegowych: dwupunktową i trójpunktową. Różnią się one pod względem elementów pokrycia przypadających na każdą ze zidentyfikowanych wartości brzegowych, które należy wykonać, aby osiągnąć 100% pokrycia.

W dwupunktowej analizie wartości brzegowych (Craig 2002, Myers 2011) dla każdej wartości brzegowej istnieją dwa elementy pokrycia: ta wartość brzegowa oraz jej najbliższy sąsiad należący do sąsiedniej klasy równoważności. Aby osiągnąć 100% pokrycia za pomocą tego wariantu techniki, przypadki testowe muszą sprawdzać wszystkie elementy pokrycia, tj. wszystkie zidentyfikowane wartości brzegowe wszystkich klas równoważności. Pokrycie mierzy się jako liczbę sprawdzonych wartości brzegowych podzieloną przez całkowitą liczbę zidentyfikowanych wartości brzegowych i wyraża się je w procentach.

W trójpunktowej analizie wartości brzegowych (Koomen 2006, O'Regan 2019) dla każdej wartości brzegowej istnieją trzy elementy pokrycia: ta wartość brzegowa oraz jej obie sąsiednie wartości. Dlatego w tym wariancie techniki niektóre elementy pokrycia mogą nie być wartościami brzegowymi. Aby osiągnąć 100% pokrycia za pomocą tego wariantu techniki, przypadki testowe muszą sprawdzać wszystkie elementy pokrycia, tj. wszystkie zidentyfikowane wartości brzegowe i ich sąsiednie wartości. Pokrycie mierzy się jako liczbę sprawdzonych wartości brzegowych i ich sąsiednich wartości podzieloną przez całkowitą liczbę zidentyfikowanych wartości brzegowych i ich sąsiednich wartości i wyraża się je w procentach.

Wariant trójpunktowy jest bardziej rygorystyczny niż dwupunktowy, ponieważ może wykryć defekty pominięte przez wariant dwupunktowy. Na przykład, jeśli decyzja „if (x ≤ 10) ..." zostanie nieprawidłowo zaimplementowana jako „if (x = 10) ...", dane testowe pochodzące z wariantu dwupunktowego (x = 10, x = 11) nie wykryją tego defektu. Jednak wartość x = 9, wymagana przez wariant trójpunktowy, prawdopodobnie ją wykryje.

4.2.3. Testowanie w oparciu o tablicę decyzyjną

Tablice decyzyjne służą do testowania implementacji wymagań opisujących, w jaki sposób różne kombinacje warunków prowadzą do różnych wyników. Tablice decyzyjne są skutecznym sposobem modelowania złożonej logiki, takiej jak reguły biznesowe.

Podczas tworzenia tablic decyzyjnych definiuje się warunki i wynikające z nich akcje systemu. Warunki i akcje tworzą wiersze tablicy. Każda kolumna odpowiada regule decyzyjnej, która definiuje unikalną kombinację warunków wraz z powiązanymi akcjami. W ograniczonych tablicach decyzyjnych wszystkie wartości warunków i akcji (z wyjątkiem nieistotnych lub niemożliwych do spełnienia; patrz poniżej) są przedstawiane jako wartości logiczne (prawda lub fałsz). W uogólnionych tablicach decyzyjnych niektóre lub wszystkie warunki i akcje przyjmują wiele wartości (np. zakresy liczb, klasy równoważności, wartości dyskretne).

Notacja dla warunków jest następująca: „P" (prawda) oznacza, że warunek jest spełniony. „F" (fałsz) oznacza, że warunek nie jest spełniony. Symbol „–" oznacza, że wartość warunku nie ma znaczenia dla wyniku akcji, a „nd" („nie dotyczy") oznacza, że warunek nie występuje dla danej reguły. W przypadku akcji „X" oznacza, że akcja powinna zostać wykonana, a puste pole oznacza, że akcja nie powinna zostać wykonana. Można również stosować inne notacje.

Pełna tablica decyzyjna ma tyle kolumn, ile jest niezbędne do pokrycia każdej kombinacji warunków. Tablicę można uprościć, usuwając kolumny zawierające niemożliwe do spełnienia kombinacje warunków. Można ją również zminimalizować, łącząc kolumny, w których niektóre

© International Software Testing Qualifications Board © Polish Quality Board

warunki nie mają wpływu na wynik, w jedną kolumnę. Algorytmy minimalizacji tablicy decyzyjnej wykraczają poza zakres niniejszego sylabusa.

W testowaniu tablicy decyzyjnej elementami pokrycia są jej kolumny zawierające możliwe do spełnienia kombinacje warunków. Aby osiągnąć 100% pokrycia przy użyciu tej techniki, przypadki testowe muszą sprawdzać wszystkie te kolumny. Pokrycie mierzy się jako liczbę sprawdzonych kolumn podzieloną przez całkowitą liczbę kolumn i wyraża się je w procentach.

Zaletą testowania w oparciu o tablicę decyzyjną jest to, że zapewnia ono systematyczne podejście do zidentyfikowania wszystkich kombinacji warunków, które w innym przypadku mogłyby zostać przeoczone. Pomaga również znaleźć luki lub sprzeczności w wymaganiach. Jeśli istnieje wiele warunków, sprawdzenie wszystkich reguł decyzyjnych może być czasochłonne, ponieważ liczba reguł rośnie wykładniczo wraz z liczbą warunków. W takim przypadku, aby zmniejszyć liczbę reguł, które należy sprawdzić, można zminimalizować tablicę decyzyjną lub zastosować podejście oparte na ryzyku.

4.2.4. Testowanie przejść pomiędzy stanami

Diagram stanów umożliwia modelowanie zachowania systemu poprzez zobrazowanie jego możliwych stanów i poprawnych przejść między nimi. Przejście jest inicjowane przez zdarzenie, które może być dodatkowo kwalifikowane przez warunek dozoru. Przyjmuje się, że przejścia są natychmiastowe i mogą niekiedy powodować wykonanie przez oprogramowanie określonej akcji. Typowa notacja stosowana do oznaczania przejść ma postać „zdarzenie [warunek dozoru] / akcja", przy czym warunki dozoru i akcje można pominąć, jeśli nie istnieją lub są nieistotne z punktu widzenia testera.

Tablica stanów jest modelem równoważnym diagramowi stanów. Jej wiersze odpowiadają stanom, a kolumny – zdarzeniom (wraz z ewentualnymi warunkami dozoru). Wpisy w tablicy stanów odpowiadają przejściom i zawierają stan docelowy, a także akcje, jeśli zostały zdefiniowane. W przeciwieństwie do diagramu stanów, tablica stanów wyraźnie pokazuje przejścia niepoprawne, reprezentowane jako puste komórki w tablicy.

Przypadek testowy oparty na diagramie stanów lub tablicy stanów to sekwencja zdarzeń, która skutkuje sekwencją zmian stanów (i wykonaniem akcji, jeśli są zdefiniowane). Jeden przypadek testowy może pokrywać wiele przejść pomiędzy stanami.

Istnieje wiele kryteriów pokrycia dla testowania przejść pomiędzy stanami. Niniejszy sylabus omawia trzy z nich.

W **pokryciu wszystkich stanów** elementami pokrycia są stany. Aby osiągnąć 100% tego pokrycia, przypadki testowe muszą osiągnąć każdy stan co najmniej raz. Pokrycie mierzy się jako liczbę osiągniętych stanów podzieloną przez całkowitą liczbę stanów i wyraża się je w procentach.

W **pokryciu przejść poprawnych** (zwanym również pokryciem 0-przełączeń) elementami pokrycia są pojedyncze poprawne przejścia. Aby osiągnąć 100% tego pokrycia, przypadki testowe muszą wykonać każde poprawne przejście co najmniej raz. Pokrycie mierzy się jako liczbę wykonanych poprawnych przejść podzieloną przez całkowitą liczbę poprawnych przejść i wyraża się je w procentach.

W **pokryciu wszystkich przejść** elementami pokrycia są wszystkie przejścia pokazane w tablicy stanów. Aby osiągnąć 100% tego pokrycia, przypadki testowe muszą wykonać każde poprawne

© International Software Testing Qualifications Board © Polish Quality Board

przejście co najmniej raz i podjąć próbę wykonania każdego niepoprawnego przejścia. Przetestowanie tylko jednego niepoprawnego przejścia w jednym przypadku testowym pomaga uniknąć maskowania defektów, tj. sytuacji, w której jeden defekt uniemożliwia wykrycie innego. Pokrycie mierzy się jako iloraz liczby poprawnych i niepoprawnych przejść, które zostały wykonane lub w przypadku których podjęto próbę wykonania za pomocą przypadków testowych przez łączną liczbę poprawnych i niepoprawnych przejść i wyraża się je w procentach.

Pokrycie wszystkich stanów jest słabsze niż pokrycie przejść poprawnych, ponieważ zazwyczaj można je osiągnąć bez wykonywania wszystkich przejść. Pokrycie przejść poprawnych jest najczęściej stosowanym kryterium pokrycia. Osiągnięcie pełnego pokrycia przejść poprawnych gwarantuje pełne pokrycie wszystkich stanów. Osiągnięcie pełnego pokrycia wszystkich przejść gwarantuje zarówno pełne pokrycie wszystkich stanów, jak i pełne pokrycie przejść poprawnych i powinno być minimalnym wymaganiem dla oprogramowania krytycznego ze względów bezpieczeństwa.

4.3. Białoskrzynkowe techniki testowania

Ze względu na ich popularność i prostotę, w tym podrozdziale skupiono się na dwóch technikach testowania białoskrzynkowego związanych z kodem:

- testowanie instrukcji,
- testowanie gałęzi.

Istnieją bardziej rygorystyczne techniki testowania białoskrzynkowego, które są stosowane w przypadku systemów krytycznych ze względów bezpieczeństwa, w systemach kluczowych dla działalności przedsiębiorstwa oraz w środowiskach wymagających wysokiego poziomu integralności w celu uzyskania pełniejszego pokrycia kodu. Istnieją również techniki testowania białoskrzynkowego stosowane na wyższych poziomach testów (np. testowanie API) lub wykorzystujące pokrycie niezwiązane z kodem (np. pokrycie neuronów w testowaniu sieci neuronowych). Techniki te nie są omawiane w niniejszym sylabusie.

4.3.1. Testowanie instrukcji i pokrycie instrukcji kodu

W testowaniu instrukcji elementami pokrycia są instrukcje wykonywalne. Celem tej techniki jest zaprojektowanie przypadków testowych, które sprawdzają instrukcje w kodzie aż do osiągnięcia akceptowalnego poziomu pokrycia. Pokrycie mierzy się jako liczbę instrukcji sprawdzonych przez przypadki testowe podzieloną przez łączną liczbę instrukcji wykonywalnych w kodzie i wyraża się je w procentach.

Osiągnięcie 100% pokrycia instrukcji gwarantuje, że każda instrukcja wykonywalna została sprawdzona co najmniej raz. W szczególności oznacza to, że każda instrukcja zawierająca defekt zostanie wykonana, co może spowodować awarię wskazującą na obecność defektu. Jednak sprawdzenie instrukcji za pomocą przypadku testowego nie zawsze wywoła awarię, na przykład może nie wykryć defektów zależnych od danych (np. dzielenie przez zero, które kończy się niepowodzeniem tylko wtedy, gdy mianownik jest zerem). Ponadto uzyskanie 100% pokrycia instrukcji nie gwarantuje, że cała logika decyzyjna została przetestowana, ponieważ testy mogą nie sprawdzić wszystkich gałęzi w kodzie (patrz sekcja 4.3.2).

© International Software Testing Qualifications Board © Polish Quality Board

4.3.2. Testowanie gałęzi i pokrycie gałęzi

Gałąź to przepływ sterowania między dwoma wierzchołkami w grafie przepływu sterowania, modelującym możliwe sekwencje wykonywania instrukcji kodu w przedmiocie testów[1]. Każdy przepływ sterowania może być bezwarunkowy (kod liniowy) lub warunkowy (tj. wynik decyzji).

W testowaniu gałęzi elementami pokrycia są gałęzie, a celem jest zaprojektowanie przypadków testowych sprawdzających gałęzie w kodzie do momentu osiągnięcia akceptowalnego poziomu pokrycia. Pokrycie mierzy się jako liczbę gałęzi sprawdzonych przez przypadki testowe podzieloną przez całkowitą liczbę gałęzi w kodzie i wyraża się je w procentach.

Uzyskanie 100% pokrycia gałęzi oznacza, że każda gałąź w kodzie, zarówno bezwarunkowa jak i warunkowa, została sprawdzona przez przypadki testowe. Gałęzie warunkowe zazwyczaj odpowiadają wynikowi („prawda" lub „fałsz") decyzji *if-then*, wynikowi z instrukcji *switch-case* lub wynikowi decyzji o wyjściu lub kontynuowaniu pętli. Jednak sprawdzenie gałęzi za pomocą przypadku testowego nie zawsze wykryje defekt (np. może nie wykryć defektów, których wystąpienie wymaga wykonania określonej ścieżki w kodzie).

Pokrycie gałęzi subsumuje pokrycie instrukcji. Oznacza to, że każdy zbiór przypadków testowych osiągający 100% pokrycia gałęzi osiąga również 100% pokrycia instrukcji (ale nie odwrotnie).

4.3.3. Korzyści wynikające z testowania białoskrzynkowego

Podstawową zaletą wszystkich białoskrzynkowych technik testowania jest to, że podczas testowania brana jest pod uwagę cała implementacja oprogramowania, co ułatwia wykrywanie defektów nawet wtedy, gdy specyfikacja oprogramowania jest niejasna, nieaktualna lub niekompletna. Wadą z kolei jest to, że jeśli oprogramowanie nie implementuje jakiegoś wymagania, testowanie białoskrzynkowe może nie wykryć tego pominięcia (Watson 1996).

Techniki białoskrzynkowe mogą być stosowane w testowaniu statycznym (np. podczas próbnych przebiegów kodu, ang. *dry runs*). Nadają się również do przeglądów kodu, który nie jest jeszcze gotowy do uruchomienia (Hetzel 1988), pseudokodu oraz innych mechanizmów logicznych, które można modelować za pomocą diagramów przepływu (np. takich jak diagram aktywności).

Wykonanie jedynie testowania czarnoskrzynkowego nie pozwala zmierzyć rzeczywistego pokrycia kodu. Miary pokrycia stosowane w technikach białoskrzynkowych zapewniają obiektywny pomiar pokrycia oraz informacje niezbędne do wygenerowania dodatkowych testów w celu zwiększenia tego pokrycia, a tym samym zwiększenia zaufania do kodu.

4.4. Techniki testowania oparte na doświadczeniu

Powszechnie stosowane techniki testowania oparte na doświadczeniu omówione w poniższych sekcjach to:

- zgadywanie błędów (ang. *error guessing*),
- testowanie eksploracyjne,
- testowanie oparte na liście kontrolnej.

[1] Zazwyczaj przyjmuje się, że wierzchołek w grafie przepływu sterowania reprezentuje nie pojedynczą instrukcję wykonywalną, lecz tzw. blok podstawowy kodu (czyli liniową sekwencję instrukcji, które zawsze muszą być wykonane albo w całości albo w ogóle) [przyp. tłum.].

© International Software Testing Qualifications Board © Polish Quality Board

4.4.1. Zgadywanie błędów

Zgadywanie błędów to technika testowania stosowana w celu przewidzenia wystąpienia błędów, defektów i awarii w oparciu o wiedzę testera, dotyczącą w szczególności:

- sposobu działania aplikacji w przeszłości,
- typów błędów popełnianych przez programistów i wynikających z nich typów defektów,
- rodzajów awarii, które wystąpiły w innych, podobnych aplikacjach.

Błędy, defekty i awarie mogą być związane z: danymi wejściowymi (np. brak akceptacji poprawnych danych wejściowych, błędne lub brakujące parametry), danymi wyjściowymi (np. nieprawidłowy format lub wynik), logiką (np. brakujące przypadki, nieprawidłowy operator), obliczeniami (np. nieprawidłowy operand, błędne obliczenie), interfejsami (np. niezgodność parametrów, niekompatybilny typ) lub danymi (np. błędna inicjalizacja, niewłaściwy typ).

Atak usterek (ang. *fault attack*) to metodyczne podejście do zgadywania błędów. Wymaga on od testera stworzenia lub uzyskania listy możliwych błędów, defektów i awarii oraz zaprojektowania testów, które zidentyfikują defekty związane z błędami, ujawnią defekty lub spowodują awarie z listy. Listy te można tworzyć w oparciu o doświadczenie, dane dotyczące defektów i awarii lub powszechną wiedzę na temat przyczyn awarii oprogramowania (np. taksonomie defektów).

Więcej informacji na temat zgadywania błędów i ataku usterek można znaleźć w (Whittaker 2002, Whittaker 2003, Andrews 2006).

4.4.2. Testowanie eksploracyjne

W testowaniu eksploracyjnym testy są jednocześnie projektowane, wykonywane i oceniane, podczas gdy tester poznaje przedmiot testów. Testowanie to służy do uzyskania większej wiedzy na temat przedmiotu testów, głębszego zbadania go za pomocą ukierunkowanych testów oraz stworzenia testów dla obszarów, które nie zostały jeszcze przetestowane.

Testowanie eksploracyjne może być wykonywane jako tzw. testowanie w sesjach, w celu ustrukturyzowania całego procesu testowania. W podejściu tym testowanie eksploracyjne jest wykonywane w określonym przedziale czasowym. Tester prowadzi testy korzystając z karty opisu testu (ang. *test charter*) zawierającej cele testów. Po zakończeniu sesji testowej zazwyczaj odbywa się spotkanie podsumowujące, które obejmuje dyskusję między testerem a interesariuszami zainteresowanymi wynikami sesji testowej. W tym podejściu cele testów mogą być traktowane jako warunki testowe wysokiego poziomu. Elementy pokrycia są identyfikowane i sprawdzane podczas sesji testowej. Tester może korzystać z arkuszy sesji testowej, aby dokumentować wykonane czynności i dokonane odkrycia.

Testowanie eksploracyjne jest przydatne, gdy specyfikacje są niepełne lub nieodpowiednie lub gdy testowanie podlega znacznej presji czasowej. Testowanie eksploracyjne jest również przydatne jako uzupełnienie innych, bardziej formalnych technik testowania. Testowanie eksploracyjne będzie bardziej skuteczne, jeśli tester ma doświadczenie, posiada wiedzę dziedzinową i wysoki poziom niezbędnych umiejętności, takich jak umiejętności analityczne, ciekawość i kreatywność (patrz sekcja 1.5.1).

© International Software Testing Qualifications Board © Polish Quality Board

Testowanie eksploracyjne może obejmować wykorzystanie innych technik testowania (np. podział na klasy równoważności). Więcej informacji na temat testowania eksploracyjnego można znaleźć w (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3. Testowanie w oparciu o listę kontrolną

W testowaniu w oparciu o listę kontrolną tester projektuje, implementuje i wykonuje testy w celu pokrycia warunków testowych z listy kontrolnej. Listy kontrolne mogą być tworzone w oparciu o doświadczenie, znajomość oczekiwań użytkownika lub wiedzę na temat przyczyn i objawów awarii oprogramowania. Listy kontrolne nie powinny zawierać elementów, które można sprawdzić automatycznie, elementów lepiej nadających się jako kryteria wejścia bądź kryteria wyjścia lub elementów, które są sformułowane w zbyt ogólny sposób (Brykczynski 1999).

Elementy listy kontrolnej są często sformułowane w formie pytań. Każdy element powinien być możliwy do sprawdzenia oddzielnie i bezpośrednio. Elementy mogą odnosić się do wymagań, właściwości interfejsu graficznego, charakterystyk jakościowych lub innych form warunków testowych. Listy kontrolne mogą być tworzone w celu wsparcia różnych typów testów, w tym testów funkcjonalnych i niefunkcjonalnych (np. 10 heurystyk użyteczności (Nielsen 1994)).

Niektóre elementy listy kontrolnej mogą z czasem stopniowo tracić na skuteczności, ponieważ programiści nauczą się unikać popełniania tych samych błędów. Konieczne może być również dodanie nowych elementów, aby odzwierciedlić nowo wykryte defekty o wysokim stopniu krytyczności. Dlatego listy kontrolne powinny być regularnie aktualizowane na podstawie analizy defektów. Należy jednak uważać, aby lista kontrolna nie stała się zbyt obszerna (Gawande 2009).

W przypadku braku szczegółowych przypadków testowych testowanie w oparciu o listy kontrolne zapewnia niezbędne wytyczne i pewien stopień spójności testowania. Jeśli listy kontrolne są wysokopoziomowe, prawdopodobne jest wystąpienie pewnej zmienności w rzeczywistym testowaniu, co może skutkować większym pokryciem, ale mniejszą powtarzalnością testów.

4.5. Podejścia do testowania oparte na współpracy

Każda z technik opisanych w podrozdziałach 4.2, 4.3 i 4.4 ma określony cel związany z wykrywaniem defektów. Podejścia do testowania oparte na współpracy koncentrują się również na unikaniu defektów poprzez zapewnienie współpracy i wymianę informacji.

4.5.1. Wspólne pisanie historyjek użytkownika

Historyjka użytkownika (ang. *user story*) reprezentuje cechę systemu cenną dla użytkownika lub nabywcy. Historyjki użytkowników mają trzy kluczowe aspekty (Jeffries 2000), tzw. „3C”:

- karta (ang. *card*) – nośnik opisujący historyjkę użytkownika (np. karta katalogowa, wpis w kartotece elektronicznej),
- rozmowa (ang. *conversation*) – wyjaśnia, w jaki sposób oprogramowanie będzie używane (może być udokumentowana lub ustna),
- potwierdzenie (ang. *confirmation*) – czyli kryteria akceptacji (patrz sekcja 4.5.2).

Najpopularniejszym formatem historyjki użytkownika jest „Jako [rola], chcę [cel do osiągnięcia], aby móc [wynikająca z tego wartość biznesowa dla roli]”, po czym następują kryteria akceptacji.

© International Software Testing Qualifications Board © Polish Quality Board

Wspólne tworzenie historyjki użytkownika może wykorzystywać techniki takie jak burza mózgów i tworzenie map myśli. Współpraca pozwala zespołowi uzyskać wspólną wizję tego, co powinno zostać dostarczone, biorąc pod uwagę trzy perspektywy: biznesową, programistyczną i testową.

Dobre historyjki użytkownika powinny być niezależne, negocjowalne, wartościowe, możliwe do oszacowania, niewielkie i możliwe do przetestowania (ang. *Independent*, *Negotiable*, *Valuable*, *Estimable*, *Small*, *Testable* – INVEST). Jeśli interesariusz nie wie, jak przetestować historyjkę użytkownika, może to oznaczać, że nie jest ona wystarczająco jasna, nie odzwierciedla czegoś wartościowego dla interesariusza lub że interesariusz po prostu potrzebuje pomocy w testowaniu (Wake 2003).

4.5.2. Kryteria akceptacji

Kryteria akceptacji dla historyjki użytkownika to warunki, które muszą zostać spełnione, aby została ona zaakceptowana przez interesariuszy. Z tej perspektywy kryteria akceptacji można postrzegać jako warunki testowe, które powinny być sprawdzane podczas testów. Kryteria akceptacji są zazwyczaj wynikiem rozmowy (patrz sekcja 4.5.1).

Kryteria akceptacji służą do:

- określenia zakresu historyjki użytkownika,
- osiągnięcia konsensusu wśród interesariuszy,
- opisania zarówno pozytywnych, jak i negatywnych scenariuszy,
- stworzenia podstawy do testowania akceptacyjnego historyjek użytkownika (patrz sekcja 4.5.3),
- umożliwienia dokładnego planowania i szacowania.

Istnieje wiele sposobów zapisywania kryteriów akceptacji dla historyjki użytkownika. Dwa najpopularniejsze formaty to:

- zorientowany na scenariusze (np. format Given/When/Then stosowany w wytwarzaniu sterowanym zachowaniem, patrz sekcja 2.1.3),
- zorientowany na reguły (np. lista kontrolna w punktach lub tabelaryczna forma mapowania danych wejściowych i wyjściowych).

Większość kryteriów akceptacji można udokumentować w jednym z tych dwóch formatów. Zespół może też użyć innego formatu, o ile kryteria akceptacji są dobrze zdefiniowane i jednoznaczne.

4.5.3. Wytwarzanie sterowane testami akceptacyjnymi (ATDD)

Wytwarzanie sterowane testami akceptacyjnymi to podejście typu „najpierw test” (patrz sekcja2.1.3), zgodnie z którym przypadki testowe są tworzone przed zaimplementowaniem historyjki użytkownika. Przypadki testowe są tworzone przez członków zespołu reprezentujących różne perspektywy, np. klientów, programistów i testerów (Adzic 2009). Przypadki testowe mogą być następnie wykonywane manualnie lub automatycznie.

Pierwszym krokiem jest warsztat specyfikacji, podczas którego członkowie zespołu analizują, omawiają i opisują historyjkę użytkownika oraz jej kryteria akceptacji (jeśli nie zostały jeszcze zdefiniowane). W trakcie tego procesu uzupełnia się braki, wyjaśnia się niejasności i usuwa defekty wykryte w historyjce użytkownika. Drugi krok to stworzenie przypadków testowych. Może

© International Software Testing Qualifications Board © Polish Quality Board

to zrobić cały zespół lub indywidualnie tester. Przypadki testowe tworzy się w oparciu o kryteria akceptacji i można je traktować jako przykłady działania oprogramowania. Pomaga to zespołowi w prawidłowej implementacji historyjki użytkownika.

Ponieważ przykłady i testy są tym samym, terminy te są często używane zamiennie. Podczas projektowania testów można stosować techniki testowania opisane w sekcjach 4.2, 4.3 i 4.4.

Zazwyczaj najpierw wykonuje się pozytywne przypadki testowe, czyli takie, które potwierdzają prawidłowe działanie (bez popełniania błędów i występowania wyjątków) i obejmują sekwencję czynności, w której wszystko przebiega zgodnie z oczekiwaniami. Po zakończeniu wykonywania pozytywnych przypadków testowych zespół powinien przeprowadzić testy negatywne. Na koniec zespół powinien uwzględnić niefunkcjonalne charakterystyki jakościowe (np. wydajność, użyteczność). Przypadki testowe powinny być wyrażone w sposób zrozumiały dla interesariuszy. Zazwyczaj zawierają one zdania w języku naturalnym, obejmujące niezbędne warunki wstępne (jeśli występują), dane wejściowe i warunki wyjściowe.

Przypadki testowe muszą obejmować wszystkie charakterystyki historyjki użytkownika i nie powinny wykraczać poza jej zakres. Natomiast kryteria akceptacji mogą szczegółowo opisywać niektóre kwestie opisane w historyjce użytkownika. Ponadto żadne dwa przypadki testowe nie powinny opisywać tej samej charakterystyki historyjki użytkownika.

Jeśli przypadki testowe zostaną zapisane w formacie obsługiwanym przez strukturę do testów automatycznych, programiści mogą zautomatyzować ich wykonywanie pisząc kod pomocniczy podczas implementacji funkcjonalności opisanej w historyjce użytkownika. Testy akceptacyjne stają się wówczas wykonywalnymi wymaganiami.

© International Software Testing Qualifications Board © Polish Quality Board

5. Zarządzanie czynnościami testowymi – 335 minut

Słowa kluczowe

analiza ryzyka, identyfikacja ryzyka, kontrola ryzyka, kryteria wejścia, kryteria wyjścia, kwadranty testowe, łagodzenie ryzyka, monitorowanie ryzyka, monitorowanie testów, nadzór nad testami, ocena ryzyka, piramida testów, plan testów, planowanie testów, podejście do testowania, poziom ryzyka, raport o defekcie, raport o postępie testów, ryzyko, ryzyko produktowe, ryzyko projektowe, strategia testów, sumaryczny raport z testów, testowanie oparte na ryzyku, zarządzanie defektami, zarządzanie ryzykiem

Cele nauczania dla rozdziału 5

5.1 Planowanie testów

FL-5.1.1 (K2) Omówić na przykładach cel i treść planu testów.

FL-5.1.2 (K1) Rozpoznawać, jaki jest wkład testera w planowanie iteracji i wydań.

FL-5.1.3 (K2) Porównać i zestawiać ze sobą kryteria wejścia i kryteria wyjścia.

FL-5.1.4 (K3) Obliczyć pracochłonność testowania przy użyciu technik szacowania.

FL-5.1.5 (K3) Stosować priorytetyzację przypadków testowych.

FL-5.1.6 (K1) Pamiętać pojęcia związane z piramidą testów.

FL-5.1.7 (K2) Podsumować kwadranty testowe oraz ich relację do poziomów testów i typów testów.

5.2 Zarządzanie ryzykiem

FL-5.2.1 (K1) Określać poziom ryzyka na podstawie prawdopodobieństwa ryzyka i wpływu ryzyka.

FL-5.2.2 (K2) Rozróżniać ryzyka projektowe i produktowe.

FL-5.2.3 (K2) Wyjaśnić potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testów.

FL-5.2.4 (K2) Wyjaśnić, jakie środki można podjąć w odpowiedzi na przeanalizowane ryzyka produktowe.

5.3 Monitorowanie testów, nadzór nad testami i ukończenie testów

FL-5.3.1 (K1) Pamiętać metryki stosowane w odniesieniu do testowania.

FL-5.3.2 (K2) Podsumować cele i treść raportów z testów oraz wskazać ich odbiorców.

FL-5.3.3 (K2) Omówić na przykładach sposób przekazywania informacji o statusie testowania.

5.4 Zarządzanie konfiguracją

FL-5.4.1 (K2) Podsumować, w jaki sposób zarządzanie konfiguracją wspomaga testowanie.

5.5 Zarządzanie defektami

FL-5.5.1 (K3) Sporządzić raport o defekcie.

© International Software Testing Qualifications Board © Polish Quality Board

5.1. Planowanie testów

5.1.1. Cel i treść planu testów

Plan testów opisuje cele, zasoby i procesy związane z projektem testowym. Plan testów:

- dokumentuje sposób i harmonogram realizacji celów testów,
- pomaga zapewnić, że przeprowadzone czynności testowe spełnią ustalone kryteria,
- służy jako środek komunikacji z członkami zespołu i innymi interesariuszami,
- pozwala wykazać, że testowanie będzie zgodne z istniejącą polityką testów i strategią testów (lub wyjaśnia ewentualne odstępstwa od nich).

Planowanie testów kieruje myślenie testerów i zmusza ich do zmierzenia się z przyszłymi wyzwaniami związanymi z ryzykami, harmonogramami, relacjami interpersonalnymi, narzędziami, kosztami, wysiłkiem itp. Proces przygotowywania planu testów (planowanie) pomaga przemyśleć działania niezbędne do osiągnięcia celów testów.

Typowy plan testów dokumentuje:

- kontekst testowania (np. zakres testów, cele testów, podstawa testów),
- założenia i ograniczenia projektu testowego,
- analizę interesariuszy (np. role, obowiązki, znaczenie dla testowania, potrzeby w zakresie zatrudniania i szkoleń),
- sposób komunikacji (np. forma i częstotliwość, szablony dokumentacji),
- rejestr ryzyk (np. ryzyk produktowych i projektowych),
- podejście do testowania (np. poziomy testów, typy testów, techniki testowania, testalia, kryteria wejścia i wyjścia, niezależność testowania, zbierane metryki, wymagania dotyczące danych testowych i środowiska testowego, odstępstwa od polityki testów i strategii testów obowiązujących w organizacji),
- budżet i harmonogram.

Więcej szczegółów na temat planu testów i jego zawartości można znaleźć w normie ISO/IEC/IEEE 29119-3.

5.1.2. Wkład testera w planowanie iteracji i wydań

W iteracyjnych modelach wytwarzania oprogramowania zazwyczaj występują dwa rodzaje planowania: planowanie wydania i planowanie iteracji.

Planowanie wydania obejmuje perspektywę przyszłego wydania produktu, definiuje i redefiniuje backlog projektu i może obejmować doprecyzowanie większych historyjek użytkownika poprzez ich podział na zestaw mniejszych historyjek użytkownika. Opracowany w ten sposób plan służy też jako podstawa podejścia do testowania i planu testów we wszystkich iteracjach w ramach wydania. Testerzy zaangażowani w planowanie wydania uczestniczą w pisaniu testowalnych historyjek użytkownika i kryteriów akceptacji (patrz podrozdział 4.5), uczestniczą w analizie ryzyka projektowego i produktowego (patrz podrozdział 5.2), szacują wysiłek związany z testowaniem historyjek użytkownika (patrz sekcja 5.1.4), określają podejście do testów i planują testowanie związane z danym wydaniem.

© International Software Testing Qualifications Board © Polish Quality Board

Planowanie iteracji obejmuje perspektywę pojedynczej iteracji i dotyczy backlogu tej iteracji (backlogu sprintu). Testerzy zaangażowani w planowanie iteracji uczestniczą w szczegółowej analizie ryzyka związanego z historyjkami użytkownika, określają testowalność historyjek użytkownika, dzielą historyjki użytkownika na zadania (w szczególności na zadania testowe), szacują pracochłonność wszystkich zadań testowych w iteracji oraz identyfikują i udoskonalają funkcjonalne i niefunkcjonalne aspekty przedmiotu testów.

5.1.3. Kryteria wejścia i kryteria wyjścia

Kryteria wejścia określają warunki wstępne podjęcia danego działania. Jeśli kryteria wejścia nie są spełnione, prawdopodobnie wykonywana czynność okaże się trudniejsza, bardziej czasochłonna, kosztowna i ryzykowna. Kryteria wyjścia określają, co należy osiągnąć, aby uznać czynność za zakończoną. Kryteria wejścia i wyjścia powinny być określone dla każdego poziomu testów i będą się różnić w zależności od celów testów.

Typowe kryteria wejścia obejmują: dostępność zasobów (np. ludzi, narzędzi, środowisk, danych testowych, budżetu, czasu), dostępność testaliów (np. podstawy testów, testowalnych wymagań, historyjek użytkownika, przypadków testowych) oraz początkowy poziom jakości przedmiotu testów (np. zaliczenie wszystkich testów dymnych).

Typowe kryteria wyjścia obejmują: miary staranności (np. osiągnięty poziom pokrycia, liczba nieusuniętych defektów, gęstość defektów, liczba niezaliczonych przypadków testowych) oraz kryteria binarne „tak/nie” (np. wykonanie wszystkich zaplanowanych testów, przeprowadzenie testowania statycznego, zgłoszenie wszystkich znalezionych defektów, zautomatyzowanie wszystkich testów regresji).

Wyczerpanie czasu lub przekroczenie budżetu można również uznać za poprawne kryteria wyjścia. Zakończenie testowania w takich okolicznościach, nawet jeśli inne kryteria wyjścia nie zostały spełnione, może być dopuszczalne, jeśli interesariusze przeanalizowali i zaakceptowali ryzyko związane z zakończeniem danego etapu bez dalszych testów.

W zwinnym wytwarzaniu oprogramowania kryteria wyjścia są często nazywane definicją ukończenia (ang. *Definition of Done*, DoD) i określają obiektywne metryki dla elementu, który można uznać za zakończony lub przekazać do eksploatacji. Kryteria wejścia, które historyjka użytkownika musi spełnić, aby rozpocząć działania związane z jej implementacją lub testowaniem, nazywane są definicją gotowości (ang. *Definition of Ready*, DoR).

5.1.4. Techniki szacowania

Szacowanie pracochłonności testów (ang. *test effort*) polega na przewidywaniu ilości pracy związanej z testowaniem, niezbędnej do osiągnięcia celów projektu testowego. Ważne jest, aby wyjaśnić interesariuszom, że szacunki opierają się na szeregu założeń i zawsze są obarczone błędem szacowania. Oszacowania dotyczące małych zadań są zazwyczaj dokładniejsze niż w przypadku dużych zadań. Dlatego też, szacując duże zadanie, można je najpierw rozbić na zestaw mniejszych zadań, które można oszacować indywidualnie z mniejszym błędem.

W niniejszym sylabusie opisano cztery następujące techniki szacowania.

Szacowanie na podstawie proporcji. W tej technice opartej na metrykach gromadzone są dane z poprzednich projektów realizowanych w organizacji, co pozwala uzyskać „standardowe”

© International Software Testing Qualifications Board © Polish Quality Board

stosunki współczynników dla podobnych projektów. Stosunki współczynników własnych projektów organizacji (np. zaczerpnięte z danych historycznych) są zazwyczaj najlepszym źródłem informacji wykorzystywanym w procesie szacowania. Te standardowe stosunki współczynników można następnie wykorzystać do oszacowania wysiłku testowego nowego projektu. Na przykład, jeśli w poprzednim projekcie stosunek pracochłonności wytwarzania do testowania wynosił 3:2, a w bieżącym projekcie wysiłek związany z wytwarzaniem ma wynieść 600 osobodni, pracochłonność testowania można oszacować na 400 osobodni z proporcji $\frac{3}{2} = \frac{600}{x}$.

Ekstrapolacja. W tej technice opartej na metrykach pomiary są wykonywane jak najwcześniej w bieżącym projekcie w celu zebrania niezbędnych danych. Dysponując wystarczającą liczbą obserwacji, pracochłonność wymaganą do wykonania pozostałych zadań można oszacować poprzez ekstrapolację tych danych, zwykle poprzez zastosowanie modelu matematycznego. Metoda ta dobrze się sprawdza w iteracyjnych modelach wytwarzania oprogramowania. Na przykład zespół może ekstrapolować pracochłonność testowania w nadchodzącej iteracji jako średnią pracochłonność z ostatnich trzech iteracji.

Szerokopasmowa technika delficka (ang. *Wideband Delphi*). W tej iteracyjnej technice opartej na wiedzy ekspertów, eksperci dokonują szacunków opartych na doświadczeniu. Każdy ekspert szacuje pracochłonność niezależnie od innych. Wyniki są gromadzone i prezentowane, a jeśli szacunki odbiegają od uzgodnionych granic, eksperci omawiają swoje aktualne szacunki. Następnie każdy ekspert jest proszony o indywidualne dokonanie nowego oszacowania na podstawie tych informacji zwrotnych. Proces ten jest powtarzany do momentu osiągnięcia konsensusu. Odmianą tej techniki jest poker planistyczny (ang. *Planning Poker*), powszechnie stosowany w zwinnym wytwarzaniu oprogramowania. W odmianie tej szacunki są zazwyczaj dokonywane przy użyciu kart z liczbami reprezentującymi wielkość pracochłonności.

Szacowanie trójpunktowe. W tej technice opartej na wiedzy ekspertów eksperci dokonują trzech oszacowań: najbardziej optymistycznego (a), najbardziej prawdopodobnego (m) i najbardziej pesymistycznego (b). Ostateczna ocena (E) jest ich ważoną średnią arytmetyczną. W najpopularniejszej wersji tej techniki ocena jest obliczana jako $E = \frac{a + 4 \cdot m + b}{6}$. Zaletą tej techniki jest to, że pozwala ekspertom obliczyć błąd pomiaru: $SD = \frac{b - a}{6}$. Na przykład, jeśli szacunki (w osobogodzinach) wynoszą: $a = 6$, $m = 9$ i $b = 18$, to ostateczny szacunek wynosi 10 ± 2 osobogodzin (tj. od 8 do 12 osobogodzin), ponieważ $E = \frac{6 + 4 \cdot 9 + 18}{6} = 10$, a $SD = \frac{18 - 6}{6} = 2$.

Więcej informacji na temat tych i wielu innych technik szacowania wysiłku testowego można znaleźć w (Kan 2003, Koomen 2006, Westfall 2009).

5.1.5. Priorytetyzacja przypadków testowych

Po wyspecyfikowaniu przypadków testowych i procedur testowych oraz zorganizowaniu ich w zestawy testowe, zestawy te można uporządkować w harmonogramie wykonywania testów, który określa kolejność uruchamiania testów. Przy priorytetyzacji przypadków testowych można wziąć pod uwagę różne czynniki.

© International Software Testing Qualifications Board © Polish Quality Board

Najczęściej stosowane strategie ustalania priorytetów przypadków testowych są następujące:

- **Priorytetyzacja oparta na ryzyku**, w której kolejność wykonywania testów wyznaczana jest na podstawie wyników analizy ryzyka (patrz sekcja 5.2.3). Najpierw wykonywane są przypadki testowe pokrywające najważniejsze ryzyka.
- **Priorytetyzacja oparta na pokryciu**, w której kolejność wykonywania testów wynika z pokrycia (np. pokrycia instrukcji). Najpierw wykonywane są przypadki testowe osiągające najwyższe pokrycie. W wariancie zwanym priorytetyzacją opartą na dodatkowym pokryciu najpierw wykonywany jest przypadek testowy osiągający najwyższe pokrycie, a każdy kolejny to taki, który osiąga najwyższe dodatkowe pokrycie (tzn. pokrycie obszarów dotąd niepokrytych przez wcześniej wykonane testy).
- **Priorytetyzacja oparta na wymaganiach**, gdzie kolejność wykonywania testów wyznaczana jest na podstawie priorytetów wymagań powiązanych z tymi przypadkami testowymi. Priorytety wymagań są definiowane przez interesariuszy. Najpierw wykonywane są przypadki testowe związane z wymaganiami uznanymi za najważniejsze.

W idealnej sytuacji przypadki testowe są wykonywane w kolejności wyznaczonej przez priorytety, na przykład przy użyciu jednej z wyżej wymienionych strategii priorytetyzacji. Jednak praktyka ta może nie sprawdzić się, jeśli pomiędzy przypadkami testowymi lub testowanymi przez nie funkcjami występują zależności. Jeśli przypadek testowy o wyższym priorytecie jest zależny od przypadku testowego o niższym priorytecie, przypadek testowy o niższym priorytecie musi zostać wykonany jako pierwszy.

Kolejność wykonywania testów powinna również uwzględniać dostępność zasobów. Na przykład, wymagane narzędzia testowe, środowiska testowe lub osoby mogą być dostępne tylko w określonym przedziale czasowym.

5.1.6. Piramida testów

Piramida testów to model pokazujący, że różne testy mogą mieć różną szczegółowość. Model ten wspiera zespół w automatyzacji testów i określaniu ich pracochłonności pokazując, że różne cele testów są wspierane przez różne poziomy automatyzacji testów. Warstwy piramidy testów reprezentują grupy testów. Im wyższa warstwa, tym mniejsza szczegółowość testu, niższa izolacja testu (tj. stopień zależności od innych elementów systemu) i dłuższy czas wykonania testu. Testy w dolnej warstwie są małe, odizolowane, szybkie i sprawdzają niewielką część funkcjonalności, więc zazwyczaj potrzeba ich wiele, aby osiągnąć rozsądny poziom pokrycia. Górna warstwa reprezentuje złożone, wysokopoziomowe, kompleksowe testy. Ich wykonywanie jest zazwyczaj wolniejsze niż wykonywanie testów z niższych warstw i zazwyczaj sprawdzają one dużą część funkcjonalności, więc zazwyczaj potrzeba niewielu takich testów, aby osiągnąć rozsądny poziom pokrycia. Liczba i nazwy warstw mogą się różnić. Na przykład, oryginalny model piramidy testów (Cohn 2009) definiuje trzy warstwy: „testy modułowe”, „testy usług” i „testy interfejsu użytkownika”. Inny popularny model definiuje testy jednostkowe (modułowe), testy integracyjne (testy integracji modułów) i testy kompleksowe (ang. *end-to-end*). Można również stosować inne poziomy testów (patrz sekcja 2.2.1).

5.1.7. Kwadranty testowe

Kwadranty testowe (Marick 2003, Crispin 2008) grupują poziomy testów z typami testów, czynnościami, technikami testowania i produktami pracy w kontekście zwinnego wytwarzania

© International Software Testing Qualifications Board © Polish Quality Board

oprogramowania. Model ten wspiera zarządzanie testami poprzez ich wizualizację, aby zapewnić uwzględnienie wszystkich odpowiednich typów testów i poziomów testów w cyklu wytwarzania oprogramowania oraz uświadomienie testerom, że niektóre typy testów są bardziej istotne na niektórych poziomach testów niż inne. Model ten zapewnia również sposób rozróżnienia i opisania poszczególnych rodzajów testów wszystkim interesariuszom, w tym programistom, testerom i przedstawicielom jednostek biznesowych.

W tym modelu testy mogą być zorientowane na biznes lub na technologię. Testy mogą również wspierać zespół (np. poprzez ukierunkowanie wytwarzania oprogramowania) lub krytykować produkt (np. poprzez pomiar zachowania przedmiotu testów w stosunku do oczekiwań). Wypadkowa tych dwóch punktów widzenia określa cztery kwadranty:

- **Kwadrant Q1 (zorientowany na technologię, wspierający zespół)** obejmuje testy modułowe i testy integracji modułów. Testy te powinny być zautomatyzowane i objęte procesem ciągłej integracji.
- **Kwadrant Q2 (zorientowany na biznes, wspierający zespół)** obejmuje testy funkcjonalne, przykłady, testy oparte na historyjkach użytkownika, prototypy doświadczeń użytkowników, testy API i symulacje. Testy te sprawdzają spełnienie kryteriów akceptacji i mogą być wykonywane manualnie lub automatycznie.
- **Kwadrant Q3 (zorientowany na biznes, krytykujący produkt)** obejmuje testy eksploracyjne, testy użyteczności i testy akceptacyjne użytkownika. Testy te są zorientowane na użytkownika i często wykonywane manualnie.
- **Kwadrant Q4 (zorientowany na technologię, krytykujący produkt)** obejmuje testy dymne i testy niefunkcjonalne (z wyjątkiem testów użyteczności). Testy te są często zautomatyzowane.

5.2. Zarządzanie ryzykiem

Organizacje borykają się z wieloma czynnikami wewnętrznymi i zewnętrznymi, które powodują niepewność co do tego, czy i kiedy osiągną swoje cele (ISO 31 000). Zarządzanie ryzykiem pozwala organizacjom zwiększyć prawdopodobieństwo osiągnięcia celów, poprawić jakość swoich produktów oraz zwiększyć zaufanie interesariuszy.

Główne czynności w zakresie zarządzania ryzykiem to:

- analiza ryzyka (obejmuje identyfikację ryzyka i ocenę ryzyka; zob. sekcja 5.2.3),
- kontrola ryzyka (obejmuje łagodzenie ryzyka i monitorowanie ryzyka; zob. sekcja 5.2.4).

Podejście do testów, w którym działania testowe są wybierane, priorytetyzowane i zarządzane w oparciu o analizę ryzyka i kontrolę ryzyka, nazywane jest testowaniem opartym na ryzyku.

5.2.1. Definicja i atrybuty ryzyka

Ryzyko to potencjalne zdarzenie, zagrożenie, niebezpieczeństwo lub sytuacja, której wystąpienie powoduje niekorzystny skutek. Ryzyko można scharakteryzować za pomocą dwóch czynników:

- **prawdopodobieństwo ryzyka**, czyli prawdopodobieństwo wystąpienia ryzyka (większe od zera i mniejsze niż jeden),
- **wpływ ryzyka** (szkoda), czyli konsekwencje jego wystąpienia.

© International Software Testing Qualifications Board © Polish Quality Board

Te dwa czynniki określają poziom ryzyka, który jest miarą ryzyka. Im wyższy poziom ryzyka, tym ważniejsze jest uwzględnienie w testach działań łagodzących to ryzyko.

5.2.2. Ryzyka projektowe i produktowe

W testowaniu zazwyczaj rozróżnia się dwa rodzaje ryzyka: projektowe i produktowe.

Ryzyka projektowe dotyczą zarządzania i nadzoru nad projektem. Ryzyka te obejmują:

- problemy organizacyjne (np. opóźnienia w dostawach produktów pracy, niedokładne szacunki, cięcia kosztów),
- problemy kadrowe (np. niewystarczające umiejętności, konflikty, problemy komunikacyjne, niedobór personelu),
- problemy techniczne (np. rozszerzenie zakresu projektu, słabe wsparcie narzędziowe),
- problemy związane z dostawcami (np. niedostarczenie niezbędnego elementu przez stronę trzecią, upadłość firmy wspierającej).

Wystąpienie ryzyk projektowych może mieć wpływ na harmonogram, budżet lub zakres projektu, a tym samym na zdolność projektu do osiągnięcia jego celów.

Ryzyka produktowe dotyczą charakterystyk jakościowych produktu (np. opisanych w modelu jakości ISO 25010). Przykłady ryzyk produktowych obejmują: brakującą lub nieprawidłową funkcjonalność, błędne obliczenia, błędy wykonania, niedopracowaną architekturę, nieefektywne algorytmy, zbyt długi czas reakcji, niski poziom doświadczenia użytkownika (ang. *user experience*, UX), luki w zabezpieczeniach. Wystąpienie ryzyk produktowych może skutkować różnymi negatywnymi konsekwencjami, w tym:

- niezadowoleniem użytkowników,
- utratą przychodów, zaufania, reputacji,
- szkodami wyrządzonymi stronom trzecim,
- wysokimi kosztami utrzymania, przeciążeniem działu pomocy technicznej,
- odpowiedzialnością karną,
- w skrajnych przypadkach – szkodami materialnymi, utratą zdrowia, a nawet życia.

5.2.3. Analiza ryzyka produktowego

Z punktu widzenia testowania celem analizy ryzyka produktowego jest uzyskanie wiedzy na temat tego ryzyka, aby koncentrować wysiłki testowe w sposób minimalizujący poziom ryzyka rezydualnego (pozostającego wciąż w produkcie). Analiza ryzyka produktowego powinna rozpocząć się na wczesnym etapie cyklu wytwarzania oprogramowania.

Analiza ryzyka produktowego obejmuje identyfikację ryzyka (ang. *risk identification*) i ocenę ryzyka (ang. *risk assessment*). Identyfikacja ryzyka polega na stworzeniu kompleksowej listy ryzyk. Interesariusze mogą identyfikować te ryzyka przy użyciu różnych technik, np. burzy mózgów, warsztatów, wywiadów lub diagramów przyczynowo-skutkowych. Ocena ryzyka obejmuje: klasyfikację zidentyfikowanych ryzyk, określenie prawdopodobieństwa ich wystąpienia, wpływu i poziomu ryzyka, ustalenie priorytetów oraz zaproponowanie sposobów łagodzenia tych ryzyk. Klasyfikacja pomaga w przypisaniu działań łagodzących ryzyko, ponieważ zazwyczaj ryzyka należące do tej samej kategorii można ograniczyć przy użyciu podobnego podejścia.

© International Software Testing Qualifications Board © Polish Quality Board

Ocena ryzyka może opierać się na podejściu ilościowym, jakościowym lub mieszanym. W podejściu ilościowym poziom ryzyka oblicza się jako iloczyn prawdopodobieństwa wystąpienia ryzyka i jego wpływu wyrażonych liczbowo. W podejściu jakościowym poziom ryzyka określa się za pomocą macierzy ryzyka.

Analiza ryzyka produktowego może mieć wpływ na staranność i zakres testowania. Jej wyniki są wykorzystywane do:

- określenia zakresu wykonywanych testów,
- określenia poziomów testów i typów testów, które należy wykonać,
- określenia technik testowania i zakładanego poziomu pokrycia,
- oszacowania pracochłonności testowania w odniesieniu do każdego zadania,
- ustalenia priorytetów testowania, aby jak najwcześniej wykryć krytyczne defekty,
- określenia, czy w celu zmniejszenia ryzyka można zastosować inne niż testowanie czynności łagodzące.

5.2.4. Kontrola ryzyka produktowego

Kontrola ryzyka produktowego obejmuje wszystkie środki podejmowane w odpowiedzi na zidentyfikowane i ocenione ryzyka produktowe. Kontrola ryzyka produktowego obejmuje łagodzenie ryzyka i monitorowania ryzyka. Łagodzenie ryzyka polega na wdrażaniu działań zaproponowanych podczas oceny ryzyka w celu zmniejszenia poziomu ryzyka. Celem monitorowania ryzyka jest zapewnienie skuteczności działań łagodzących ryzyko, uzyskanie dalszych informacji pozwalających usprawnić ocenę ryzyka oraz identyfikację nowych ryzyk.

Po przeanalizowaniu danego ryzyka, w ramach procesu kontroli ryzyka produktowego, może być zaproponowane podjęcie różnych działań łagodzących, np. ograniczenie ryzyka poprzez testowanie, akceptacja ryzyka, przeniesienie ryzyka lub plany awaryjne (Veenendaal 2012). Działania, które można podjąć w celu ograniczenia ryzyk produktowych poprzez testowanie to:

- wybór testerów posiadających poziom doświadczenia i umiejętności odpowiedni do danego rodzaju ryzyka,
- zastosowanie odpowiedniego poziomu niezależności testowania,
- przeprowadzenie przeglądów i analizy statycznej,
- zastosowanie odpowiednich technik testowania i poziomów pokrycia,
- zastosowanie odpowiednich typów testów uwzględniających charakterystyki jakościowe, na które ryzyko ma wpływ,
- wykonanie testowania dynamicznego, w tym testowania regresji.

5.3. Monitorowanie testów, nadzór nad testami i ukończenie testów

Monitorowanie testów polega na gromadzeniu informacji dotyczących testowania. Informacje te służą do oceny postępu testów oraz do pomiaru spełnienia kryteriów wyjścia lub wykonania zadań testowych związanych z kryteriami wyjścia (np. osiągnięcie zakładanego pokrycia ryzyk produktowych, wymagań lub kryteriów akceptacji).

Nadzór nad testami wykorzystuje informacje uzyskane podczas monitorowania testów w celu określenia, w formie tzw. dyrektyw nadzoru, niezbędnych działań korygujących, aby osiągnąć maksymalną efektywność i wydajność testowania.

© International Software Testing Qualifications Board © Polish Quality Board

Przykłady dyrektyw nadzoru obejmują:

- zmianę priorytetów testów, gdy zidentyfikowane ryzyko faktycznie wystąpi,
- ponowną ocenę, czy element testowy spełnia kryteria wejścia lub wyjścia w związku z wprowadzeniem do niego poprawek,
- modyfikację harmonogramu testów w celu uwzględnienia opóźnienia w dostarczeniu środowiska testowego,
- dodanie nowych zasobów w razie potrzeby.

Ukończenie testów polega na zebraniu danych z zakończonych czynności testowych w celu zebrania doświadczeń, testaliów i innych istotnych informacji. Czynności związane z ukończeniem testów mają miejsce w kluczowych momentach projektu, takich jak ukończenie poziomu testów, zakończenie iteracji w projekcie zwinnym, zakończenie (lub anulowanie) projektu testowego, wydanie oprogramowania lub zakończenie prac nad wydaniem pielęgnacyjnym (ang. *maintenance release*).

5.3.1. Metryki stosowane w testowaniu

Metryki testowe są gromadzone w celu pomiaru postępów w stosunku do planowanego harmonogramu i budżetu, aktualnej jakości przedmiotu testów oraz skuteczności czynności testowych w odniesieniu do celów testów lub celu iteracji. W ramach monitorowania testów gromadzi się różne metryki w celu wsparcia nadzoru nad testami i ukończenia testów.

Metryki testowe mierzą w szczególności:

- postęp projektu (np. realizacja zadań, wykorzystanie zasobów, pracochłonność),
- postęp testów (np. postęp implementacji przypadków testowych, postęp przygotowania środowiska testowego, liczba wykonanych/niewykonanych przypadków testowych, liczba zaliczonych/niezaliczonych przypadków testowych, czas wykonywania testów),
- jakość produktu (np. dostępność, czas odpowiedzi, średni czas do awarii),
- defekty (np. liczba i priorytety wykrytych/usuniętych defektów, gęstość defektów, odsetek wykrytych defektów),
- ryzyko (np. poziom ryzyka rezydualnego),
- pokrycie (np. pokrycie wymagań, pokrycie kodu),
- koszty (np. koszt testowania, koszt jakości ponoszony przez organizację).

5.3.2. Cel, treść i odbiorcy raportów z testów

Raporty z testów służą do podsumowania i przekazania informacji dotyczących testów w trakcie ich trwania i po ich zakończeniu. **Raporty o postępie testów** wspierają w sposób ciągły nadzór nad testami, więc muszą zawierać informacje niezbędne do umożliwienia wprowadzenia zmian w harmonogramie testów, zasobach lub planie testów, gdy takie zmiany są konieczne ze względu na odstępstwa od planu lub zmianę okoliczności. **Sumaryczne raporty z testów** zawierają podsumowanie konkretnej czynności testowej (np. poziomu testów, cyklu testowego, iteracji) i mogą dostarczyć informacji przydatnych do testów w kolejnych etapach.

Podczas monitorowania testów i nadzoru nad testami zespół testowy generuje raporty o postępie testów, aby zapewnić interesariuszom bieżące informacje o postępie. Raporty o postępie testów

© International Software Testing Qualifications Board © Polish Quality Board

są zazwyczaj generowane regularnie (np. codziennie, co tydzień itp.) i zawierają między innymi informacje na temat:

- okresu testowania,
- postępu testów (np. wyprzedzenie lub opóźnienie w stosunku do harmonogramu), w tym wszelkich istotnych odchyleń,
- przeszkód w testowaniu i sposobów ich obejścia,
- metryk testowych (przykłady metryk podano w sekcji 5.3.1),
- nowych i zmienionych ryzyk w okresie testowania,
- testów zaplanowanych na następny okres.

Sumaryczny raport z testów jest przygotowywany na etapie ukończenia testów, gdy projekt, poziom testów lub typ testów został zakończony i, w idealnym przypadku, spełnione zostały kryteria wyjścia. Raport ten wykorzystuje dane z raportów o postępie testów, a także inne dane. Typowy sumaryczny raport z testów zawiera między innymi informacje dotyczące:

- podsumowania testów,
- oceny testowania i jakości produktu w oparciu o pierwotny plan testów (tj. cele testów i kryteria wyjścia),
- odchyleń od planu testów (np. różnice w stosunku do planowanego harmonogramu testów, czasu trwania i pracochłonności),
- przeszkód w testowaniu i sposobach ich obejścia,
- metryk testowych określonych na podstawie raportów o postępie testów,
- niezłagodzonych ryzyk i nieusuniętych usterek,
- wniosków istotnych dla testowania.

Różne grupy odbiorców wymagają różnych informacji w raportach, co wpływa też na stopień sformalizowania oraz częstotliwość raportowania testów. Raportowanie postępu testów w obrębie zespołu jest zwykle częste i nieformalne. Raportowanie ukończenia testów odbywa się zazwyczaj według ustalonego szablonu i ma miejsce tylko raz.

Norma ISO/IEC/IEEE 29119-3 zawiera wzory i przykłady raportu o postępie testów (zwanego w tej normie raportem o statusie testów) oraz sumarycznego raportu z testów.

5.3.3. Przekazywanie informacji o statusie testowania

Optymalny sposób komunikowania statusu testowania różni się w zależności od kwestii związanych z zarządzaniem testami, strategii testów, obowiązujących norm prawnych, a w przypadku samoorganizujących się zespołów (patrz sekcja 1.5.2), od samego zespołu. Dostępne opcje obejmują:

- komunikację werbalną z członkami zespołu i innymi interesariuszami,
- tablice wskaźników (ang. *dashboards*), np. kokpity menedżerskie, tablice zadań i wykresy spalania (ang. *burn-down charts*),
- kanały komunikacji elektronicznej (np. e-mail, czat),
- dokumentacja online,
- formalne raporty z testów (patrz sekcja 5.3.2).

© International Software Testing Qualifications Board © Polish Quality Board

Zależnie od sytuacji można skorzystać z jednej lub kilku z powyższych opcji. Bardziej formalna komunikacja może być bardziej odpowiednia dla zespołów rozproszonych, w których bezpośrednia komunikacja twarzą w twarz nie zawsze jest możliwa ze względu na odległość geograficzną lub różnice czasowe. Zazwyczaj różni interesariusze są zainteresowani różnymi rodzajami informacji, dlatego komunikacja powinna być zawsze odpowiednio dostosowana.

5.4. Zarządzanie konfiguracją

W kontekście testowania zarządzanie konfiguracją to proces zapewniający identyfikację, kontrolę i śledzenie produktów pracy, takich jak plany testów, strategie testów, warunki testowe, przypadki testowe, skrypty testowe, wyniki testów, dzienniki testów i raporty z testów. W ramach zarządzania konfiguracji te produkty pracy nazywane są elementami konfiguracji.

W przypadku złożonego elementu konfiguracji (np. środowiska testowego) zarządzanie konfiguracją rejestruje jego elementy składowe, ich relacje i wersje. Jeśli element konfiguracji zostanie zatwierdzony do testowania, staje się konfiguracją bazową (ang. *baseline*) i może być zmieniony tylko w ramach formalnego procesu zarządzania zmianą.

Zarządzanie konfiguracją przechowuje zapis zmienionych elementów konfiguracji po utworzeniu nowej konfiguracji bazowej. Dzięki temu możliwy jest powrót do poprzedniej konfiguracji bazowej w celu odtworzenia poprzednich wyników testów.

Zarządzanie konfiguracją wspiera sprawny przebieg testowania poprzez zapewnienie, że:

- wszystkie elementy konfiguracji, w tym elementy testowe (i poszczególne elementy przedmiotu testów), są jednoznacznie identyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane z innymi elementami konfiguracji, tak aby można było zachować możliwość śledzenia w całym procesie testowania,
- wszystkie zidentyfikowane dokumenty i elementy oprogramowania są przywoływane w jednoznaczny sposób w testaliach.

Ciągła integracja, ciągłe dostarczanie, ciągłe wdrażanie i związane z nimi procesy testowania są zazwyczaj wdrażane w ramach zautomatyzowanego potoku dostarczania DevOps (patrz sekcja 2.1.4), w którym zazwyczaj uwzględnia się zautomatyzowane zarządzanie konfiguracją.

5.5. Zarządzanie defektami

Ponieważ jednym z głównych celów testowania jest wykrywanie defektów, niezbędne jest ustanowienie procesu zarządzania defektami. Chociaż mówimy tutaj o „defektach”, zgłaszane anomalie mogą okazać się rzeczywistymi defektami lub czymś innym (np. wynikiem fałszywie pozytywnym lub żądaniem zmiany). Kwestia ta jest rozstrzygana podczas procesu rozpatrywania raportów o defektach. Anomalie mogą być zgłaszane na każdym etapie cyklu wytwarzania oprogramowania, a sposób zgłoszenia zależy od tego cyklu. Proces zarządzania defektami obejmuje co najmniej przepływ pracy związany z obsługą poszczególnych defektów lub anomalii od momentu ich wykrycia do zamknięcia zgłoszenia oraz reguły klasyfikacji anomalii. Przepływ pracy zazwyczaj obejmuje czynności związane z rejestrowaniem zgłoszonych anomalii, analizowaniem i klasyfikowaniem ich, podejmowaniem decyzji dotyczących odpowiedniej reakcji (takiej jak np. usunięcie lub pozostawienie stanu obecnego), a na koniec zamknięciem zgłoszenia defektu. Proces ten musi być przestrzegany przez wszystkich interesariuszy. Zaleca się podobne

© International Software Testing Qualifications Board © Polish Quality Board

postępowanie w przypadku defektów wykrytych podczas testowania statycznego (zwłaszcza analizy statycznej).

Typowy raport o defekcie ma na celu:

- dostarczenie osobom odpowiedzialnym za obsługę i rozwiązywanie zgłoszonych defektów wystarczających informacji do rozwiązania problemu,
- zapewnienie środków umożliwiających śledzenie jakości produktu pracy,
- dostarczenie pomysłów na ulepszenie procesu wytwarzania i testowania.

Raport o defekcie zarejestrowany podczas testowania dynamicznego zazwyczaj zawiera:

- unikalny identyfikator,
- tytuł i krótkie podsumowanie zgłaszanej anomalii,
- datę zaobserwowania anomalii, organizację zgłaszającą i autora, w tym jego rolę,
- identyfikację przedmiotu testów i środowiska testowego,
- kontekst wystąpienia defektu (np. uruchomiony przypadek testowy, wykonywana czynność testowa, faza cyklu wytwarzania oprogramowania oraz inne istotne informacje, takie jak technika testowania, wykorzystana lista kontrolna lub użyte dane testowe),
- opis awarii umożliwiający jej odtworzenie i usunięcie defektu, w tym kroki testowe, które doprowadziły do wystąpienia anomalii, wszelkie istotne dzienniki testów, zrzuty baz danych, zrzuty ekranu lub nagrania,
- oczekiwane wyniki i rzeczywiste wyniki,
- krytyczność defektu, tzn. stopień wpływu na interesariuszy lub wymagania,
- priorytet naprawy (pilność),
- status defektu (np.: otwarty, odroczony, duplikat, oczekujący na poprawkę, oczekujący na testowanie potwierdzające, ponownie otwarty, zamknięty, odrzucony),
- istotne odwołania do innych elementów (np. do przypadku testowego).

Niektóre z powyższych informacji mogą być automatycznie uwzględnione podczas korzystania z narzędzi do zarządzania defektami (np. identyfikator, data, autor, początkowy status). Szablon raportu o defekcie oraz przykładowe raporty o defektach można znaleźć w normie ISO/IEC/IEEE 29119-3, w której raport o defekcie nazywany jest raportem o incydencie.

© International Software Testing Qualifications Board © Polish Quality Board

6. Narzędzia testowe – 20 minut

Słowa kluczowe

automatyzacja testów

Cele nauczania dla rozdziału 5

6.1 Narzędzia wspomagające testowanie

FL-6.1.1 (K2) Wyjaśnić, w jaki sposób różnego typu narzędzia testowe wspomagają testowanie.

6.2 Korzyści i ryzyka związane z automatyzacją testów

FL-6.2.1 (K1) Pamiętać korzyści i ryzyka związane z automatyzacją testów.

© International Software Testing Qualifications Board © Polish Quality Board

6.1. Narzędzia wspomagające testowanie

Narzędzia testowe wspierają i ułatwiają wiele czynności testowych. Przykłady obejmują:

- narzędzia do zarządzania testami – zwiększają wydajność procesu testowego, ułatwiając zarządzanie cyklem wytwarzania oprogramowania, wymaganiami, testami, defektami i konfiguracją,
- narzędzia do testowania statycznego – wspierają w przeprowadzaniu przeglądów i analizy statycznej,
- narzędzia do projektowania i implementacji testów – ułatwiają tworzenie przypadków testowych, danych testowych i procedur testowych,
- narzędzia do wykonywania testów i pomiaru pokrycia – ułatwiają automatyczne wykonywanie testów i pomiar pokrycia,
- narzędzia do testowania niefunkcjonalnego – umożliwiają przeprowadzanie testów niefunkcjonalnych, które są trudne lub niemożliwe do wykonania manualnie,
- narzędzia DevOps – wspierają działanie potoku dostarczania DevOps, śledzenie przepływu pracy, zautomatyzowane procesy kompilacji, funkcjonowanie mechanizmów ciągłej integracji i ciągłego dostarczania,
- narzędzia do współpracy – ułatwiają komunikację,
- narzędzia wspierające skalowalność i standaryzację wdrażania (np. maszyny wirtualne, narzędzia do konteneryzacji),
- wszelkie inne narzędzia wspomagające testowanie (np. w kontekście testowania narzędziem testowym może być arkusz kalkulacyjny).

6.2. Korzyści i ryzyka związane z automatyzacją testów

Samo nabycie narzędzia nie gwarantuje sukcesu. Każde nowe narzędzie wymaga wysiłku, aby osiągnąć rzeczywiste i trwałe korzyści z jego użycia (np. wdrożenie narzędzia, konserwacja i szkolenia). Należy również uwzględnić pewne ryzyka, które wymagają analizy i łagodzenia.

Potencjalne korzyści wynikające z automatyzacji testów obejmują:

- oszczędność czasu dzięki ograniczeniu powtarzalnych czynności manualnych (np. wykonywanie testów regresji, ponowne wprowadzanie tych samych danych testowych, porównywanie oczekiwanych wyników z rzeczywistymi wynikami oraz sprawdzanie zgodności z normami kodowania),
- zapobieganie prostym błędom ludzkim dzięki większej spójności i powtarzalności (np. poprzez wyprowadzanie testów w spójny sposób z wymagań, systematyczne tworzenie danych testowych, wykonywanie testów przez narzędzie w tej samej kolejności i z tą samą częstotliwością),
- bardziej obiektywną ocenę (np. w zakresie pokrycia) i wyliczanie miar, które są zbyt skomplikowane, aby mogły być określone ręcznie,
- łatwiejszy dostęp do informacji o testowaniu, wspierający zarządzanie testami i raportowanie (np. statystyk, wykresów i zagregowanych danych dotyczących postępu testów, częstości awarii, czasu trwania testów),

© International Software Testing Qualifications Board © Polish Quality Board

- skrócenie czasu wykonywania testów, co skutkuje wcześniejszym wykrywaniem defektów, szybszym przekazywaniem informacji zwrotnych i skróceniem czasu wprowadzenia produktu na rynek,
- więcej czasu dla testerów na projektowanie nowych, bardziej szczegółowych i skuteczniejszych testów.

Potencjalne ryzyka związane z automatyzacją testów obejmują:

- nierealistyczne oczekiwania dotyczące korzyści płynących z narzędzia (w tym funkcjonalności i łatwości obsługi),
- niedokładne oszacowanie czasu, kosztów i pracochłonności wymaganych do wdrożenia narzędzia, utrzymania skryptów testowych i zmiany istniejącego procesu testowania manualnego,
- korzystanie z narzędzia testowego, gdy bardziej odpowiednie jest testowanie manualne,
- zbytnie poleganie na narzędziu, np. ignorowanie potrzeby krytycznego myślenia,
- uzależnienie od dostawcy narzędzia, który może zakończyć działalność, wycofać narzędzie z rynku, sprzedać je innemu dostawcy lub świadczyć wsparcie niskiej jakości (np. w zakresie reakcji na zapytania, dostarczania aktualizacji i usuwania defektów),
- korzystanie z oprogramowania o otwartym kodzie źródłowym (ang. *open-source*), które może zostać porzucone, co oznacza, że nie będą dostępne dalsze aktualizacje lub wewnętrzne moduły narzędzia będą wymagać dość częstych aktualizacji w ramach dalszego rozwoju,
- niekompatybilność narzędzia z używaną w organizacji platformą programistyczną,
- wybór nieodpowiedniego narzędzia, które nie spełnia wymogów prawnych, regulacyjnych lub norm w zakresie bezpieczeństwa.

© International Software Testing Qualifications Board © Polish Quality Board

7. Bibliografia

Normy

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering – Software testing – Part 4: Test techniques

ISO/IEC 25010, (2023-11) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality models

ISO/IEC 20246 (2017) Software and systems engineering – Work product reviews

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Książki

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P., Offutt, J. (2016) Introduction to Software Testing (wyd. 2), Cambridge University Press

Andrews, M., Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

Beizer, B. (1990) Software Testing Techniques (wyd. 2), Van Nostrand Reinhold: Boston, MA

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J. N., Randell B. (red.) (1970) Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 października 1969, s. 16

Chelimsky, D. i in. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA

Craig, R., Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood, MA

Crispin, L., Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston, MA

© International Software Testing Qualifications Board © Polish Quality Board

Forgács, I., Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande, A. (2009) The Checklist Manifesto: How to Get Things Right, Metropolitan Books: New York, NY

Gärtner, M. (2011) ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston, MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison-Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing (wyd. 2), John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (wyd. 4), CRC Press: Boca Raton, FL

Kan, S. (2003) Metrics and Models in Software Quality Engineering (wyd. 2), Addison-Wesley, polskie wydanie: (2006) Metryki i modele w inżynierii jakości oprogramowania, Mikom

Kaner, C., Falk, J., Nguyen, H. Q. (1999) Testing Computer Software (wyd. 2), Wiley Computer Publishing

Kaner, C., Bach, J., Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach (wyd. 1), Wiley Computer Publishing

Kim, G., Humble, J., Debois, P., Willis, J. (2016) The DevOps Handbook, IT Revolution Press: Portland, OR

Koomen, T., van der Aalst, L., Broekman, B., Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) The Art of Software Testing (wyd. 3), John Wiley & Sons: New York, NY, polskie wydanie: (2005) Sztuka testowania oprogramowania, Helion

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R. S. (2019) Software Engineering. A Practitioner's Approach (wyd. 9), McGraw Hill, polskie wydanie: (2010) Praktyczne podejście do inżynierii oprogramowania, WNT

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E. (red.) (2012) Practical Risk-Based Testing: The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A. H., Wallace, D. R., McCabe, T. J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

© International Software Testing Qualifications Board © Polish Quality Board

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison-Wesley

Whittaker, J., Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison Wesley Professional

Artykuły i strony internetowe

Brykczynski, B. (1999) „A survey of software inspection checklists”, *ACM SIGSOFT Software Engineering Notes* 24(1), s. 82–89

Enders, A. (1975) „An Analysis of Errors and Their Causes in System Programs”, *IEEE Transactions on Software Engineering* 1(2), s. 140–149

Manna, Z., Waldinger, R. (1978) “The logic of computer programming”, *IEEE Transactions on Software Engineering* 4(3), s. 199–229

Marick, B. (2003) „Exploration through Example”, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) „Enhancing the explanatory power of usability heuristics”, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press*, s. 152–158

Salman. I. (2016) „Cognitive biases in software quality and testing”, *Proceedings of the 38th International Conference on Software Engineering Companion* (ICSE '16), ACM Press, s. 823–826

Wake, B. (2003) „INVEST in Good Stories, and SMART Tasks”, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

© International Software Testing Qualifications Board © Polish Quality Board

8. Dodatek A – cele nauczania i poziomy wiedzy

Poniżej przedstawiono cele nauczania obowiązujące w przypadku niniejszego sylabusa. Znajomość każdego z zagadnień poruszonych w sylabusie będzie sprawdzana na egzaminie zgodnie z przypisanym celem nauczania. Opis celu nauczania zawiera czasownik wyrażający czynność, która odpowiada właściwemu poziomowi wiedzy wymienionemu poniżej.

Poziom 1 – zapamiętać (K1): kandydat pamięta, rozpoznaje lub umie sobie przypomnieć dany termin lub dane pojęcie.

Czasownik wyrażający czynność: pamiętać, rozpoznać, określić, wskazać.

Przykłady:

„Wskazać typowe cele testów".
„Pamiętać pojęcia związane z piramidą testów".
„Rozpoznawać, jaki jest wkład testera w planowanie iteracji i wydań".

Poziom 2 – zrozumieć (K2): kandydat potrafi uzasadnić lub wyjaśnić stwierdzenia dotyczące danego zagadnienia, a także podsumować, porównać i sklasyfikować pojęcia z zakresu testowania oraz podać odpowiednie przykłady.

Czasownik wyrażający czynność: sklasyfikować, porównać, zestawić ze sobą, objaśnić, omówić, rozróżnić, odróżnić, wyjaśnić, podać przykłady, podsumować.

Przykłady:

„Klasyfikować różne sposoby pisania kryteriów akceptacji".
„Porównać poszczególne role występujące w testowaniu".
„Rozróżniać ryzyka projektowe i produktowe" (pozwala na rozróżnienie koncepcji).
„Omówić na przykładach cel i treść planu testów".
„Wyjaśnić wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie".
„Podsumować czynności wykonywane w ramach procesu przeglądu".

Poziom 3 – zastosować (K3): kandydat potrafi wykonać odpowiednią procedurę, gdy zostanie mu postawione znane mu zadanie, bądź wybrać właściwą procedurę i zastosować ją w danym kontekście.

Czasownik wyrażający czynność: zastosować, obliczyć, sporządzić, użyć.

Przykłady:

„Stosować priorytetyzację przypadków testowych" (powinno odnosić się do procedury, techniki, procesu, algorytmu etc.).
„Sporządzić raport o defekcie".
„Zastosować technikę analizy wartości brzegowych, aby zaprojektować przypadki testowe".

Materiały dodatkowe w zakresie poziomów poznawczych celów nauczania:

Anderson, L. W., Krathwohl, D. R. (red.) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon.

© International Software Testing Qualifications Board © Polish Quality Board

9. Dodatek B – macierz powiązań między celami biznesowymi a celami nauczania

Cele nauczania	Poziom K	BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8	BO9	BO10	BO11	BO12	BO13	BO14
1. Podstawy testowania															
1.1 Co to jest testowanie?															
FL-1.1.1 Wskazać typowe cele testów.	K1	X													
FL-1.1.2 Odróżniać testowanie od debugowania.	K2		X												
1.2 Dlaczego testowanie jest niezbędne?															
FL-1.2.1 Podać przykłady wskazujące, dlaczego testowanie jest niezbędne.	K2	X													
FL-1.2.2 Pamiętać, jaka jest relacja między testowaniem a zapewnieniem jakości.	K1		X												
FL-1.2.3 Odróżniać podstawową przyczynę, pomyłkę, defekt i awarię.	K2		X												
1.3 Zasady testowania															
FL-1.3.1 Objaśnić siedem zasad testowania.	K2		X												
1.4 Czynności testowe, testalia i role związane z testami															
FL-1.4.1 Wyjaśnić poszczególne czynności testowe i powiązane z nimi zadania.	K2			X											
FL-1.4.2 Wyjaśnić wpływ kontekstu na proces testowy.	K2			X			X								
FL-1.4.3 Rozróżniać testalia wspomagające czynności testowe.	K2			X											
FL-1.4.4 Wyjaśnić korzyści wynikające ze śledzenia powiązań.	K2				X	X									
FL-1.4.5 Porównać poszczególne role występujące w testowaniu.	K2										X				
1.5 Niezbędne umiejętności i dobre praktyki w dziedzinie testowania															
FL-1.5.1 Podać przykłady ogólnych umiejętności wymaganych w testowaniu.	K2												X		
FL-1.5.2 Pamiętać, jakie są zalety podejścia „cały zespół”.	K1										X				
FL-1.5.3 Odróżniać korzyści i wady niezależności testowania.	K2			X											
2. Testowanie w cyklu wytwarzania oprogramowania															
2.1 Testowanie w kontekście cyklu wytwarzania oprogramowania															
FL-2.1.1 Wyjaśnić wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie.	K2						X								
FL-2.1.2 Pamiętać dobre praktyki testowania mające zastosowanie do wszystkich modeli cyklu wytwarzania oprogramowania.	K1						X								

© International Software Testing Qualifications Board © Polish Quality Board

Cele nauczania	Poziom K	BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8	BO9	BO10	BO11	BO12	BO13	BO14
FL–2.1.3 Podać przykłady podejść typu „najpierw test” w kontekście wytwarzania oprogramowania.	K1					X									
FL–2.1.4 Podsumować, w jaki sposób metodyka DevOps może wpłynąć na testowanie.	K2					X	X			X	X				
FL–2.1.5 Wyjaśnić, na czym polega przesunięcie w lewo.	K2					X	X								
FL–2.1.6 Wyjaśnić, w jaki sposób retrospektywy mogą posłużyć jako mechanizmy doskonalenia procesów.	K2					X					X				
2.2 Poziomy testów i typy testów															
FL–2.2.1 Rozróżniać poszczególne poziomy testów.	K2		X	X											
FL–2.2.2 Rozróżniać poszczególne typy testów.	K2		X												
FL–2.2.3 Odróżniać testowanie potwierdzające od testowania regresji.	K2		X												
2.3 Testowanie pielęgnacyjne															
FL–2.3.1 Podsumować testowanie pielęgnacyjne i zdarzenia je wyzwalające.	K2		X					X							
3. Testowanie statyczne															
3.1 Podstawy testowania statycznego															
FL-3.1.1 Rozpoznawać typy produktów pracy, które mogą być badane za pomocą testowania statycznego.	K1				X	X									
FL-3.1.2 Wyjaśnić korzyści wynikające z testowania statycznego.	K2	X			X	X									
FL-3.1.3 Porównać i zestawiać ze sobą testowanie statyczne i testowanie dynamiczne.	K2				X	X									
3.2 Informacje zwrotne i proces przeglądu															
FL-3.2.1 Pamiętać korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy.	K1	X			X						X				
FL-3.2.2 Podsumować czynności wykonywane w ramach procesu przeglądu.	K2			X	X										
FL-3.2.3 Pamiętać, jakie obowiązki są przypisane do najważniejszych ról w trakcie wykonywania przeglądów.	K1				X								X		
FL-3.2.4 Porównać i zestawiać ze sobą różne typy przeglądów.	K2		X												
FL-3.2.5 Pamiętać, jakie czynniki decydują o powodzeniu przeglądu.	K1					X							X		
4. Analiza i projektowanie testów															
4.1 Ogólna charakterystyka technik testowania															
FL-4.1.1 Rozróżniać czarnoskrzynkowe techniki testowania, białoskrzynkowe techniki testowania oraz techniki testowania oparte na doświadczeniu.	K2		X												
4.2 Czarnoskrzynkowe techniki testowania															

© International Software Testing Qualifications Board © Polish Quality Board

Cele nauczania	Poziom K	BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8	BO9	BO10	BO11	BO12	BO13	BO14
FL-4.2.1 Zastosować technikę podziału na klasy równoważności, aby zaprojektować przypadki testowe.	K3					X									
FL-4.2.2 Zastosować technikę analizy wartości brzegowych, aby zaprojektować przypadki testowe.	K3					X									
FL-4.2.3 Zastosować technikę testowania w oparciu o tablicę decyzyjną, aby zaprojektować przypadki testowe.	K3					X									
FL-4.2.4 Zastosować technikę testowania przejść pomiędzy stanami, aby zaprojektować przypadki testowe.	K3					X									
4.3 Białoskrzynkowe techniki testowania															
FL-4.3.1 Wyjaśnić technikę testowania instrukcji.	K2		X												
FL-4.3.2 Wyjaśnić technikę testowania gałęzi.	K2		X												
FL-4.3.3 Wyjaśnić korzyści wynikające z testowania białoskrzynkowego.	K2	X	X												
4.4 Techniki testowania oparte na doświadczeniu															
FL-4.4.1 Wyjaśnić technikę zgadywania błędów.	K2		X												
FL-4.4.2 Wyjaśnić technikę testowania eksploracyjnego.	K2		X												
FL-4.4.3 Wyjaśnić technikę testowania w oparciu o listę kontrolną.	K2		X												
4.5 Podejścia do testowania oparte na współpracy															
FL-4.5.1 Wyjaśnić, w jaki sposób należy pisać historyjki użytkownika we współpracy z programistami i przedstawicielami jednostek biznesowych.	K2				X						X				
FL-4.5.2 Klasyfikować różne sposoby pisania kryteriów akceptacji.	K2										X				
FL-4.5.3 Zastosować metodę wytwarzania sterowanego testami akceptacyjnymi (ATDD), aby zaprojektować przypadki testowe	K3					X									
5. Zarządzanie czynnościami testowymi															
5.1 Planowanie testów															
FL-5.1.1 Omówić na przykładach cel i treść planu testów.	K2		X					X							
FL-5.1.2 Rozpoznawać, jaki jest wkład testera w planowanie iteracji i wydań.	K1	X									X		X		
FL-5.1.3 Porównać i zestawiać ze sobą kryteria wejścia i kryteria wyjścia.	K2				X		X								X
FL-5.1.4 Obliczyć pracochłonność testowania przy użyciu technik szacowania.	K3							X		X					
FL-5.1.5 Stosować priorytetyzację przypadków testowych.	K3							X		X					
FL-5.1.6 Pamiętać pojęcia związane z piramidą testów.	K1		X												
FL-5.1.7 Podsumować kwadranty testowe oraz ich relację do poziomów testów i typów testów.	K2		X							X					

© International Software Testing Qualifications Board © Polish Quality Board

Cele nauczania	Poziom K	BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8	BO9	BO10	BO11	BO12	BO13	BO14
5.2 Zarządzanie ryzykiem															
FL-5.2.1 Określać poziom ryzyka na podstawie prawdopodobieństwa ryzyka i wpływu ryzyka.	K1							X						X	
FL-5.2.2 Rozróżniać ryzyka projektowe i produktowe.	K2		X											X	
FL-5.2.3 Wyjaśnić potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testów.	K2					X				X				X	
FL-5.2.4 Wyjaśnić, jakie środki można podjąć w odpowiedzi na przeanalizowane ryzyka produktowe.	K2		X			X								X	
5.3 Monitorowanie testów, nadzór nad testami i ukończenie testów															
FL-5.3.1 Pamiętać metryki stosowane w odniesieniu do testowania.	K1									X					X
FL-5.3.2 Podsumować cele i treść raportów z testów oraz wskazać ich odbiorców.	K2					X				X					X
FL-5.3.3 Omówić na przykładach sposób przekazywania informacji o statusie testowania.	K2												X		X
5.4 Zarządzanie konfiguracją															
FL-5.4.1 Podsumować, w jaki sposób zarządzanie konfiguracją wspomaga testowanie.	K2					X		X							
5.5 Zarządzanie defektami															
FL-5.5.1 Sporządzić raport o defekcie.	K3		X						X						
6. Narzędzia testowe															
6.1 Narzędzia wspomagające testowanie															
FL-6.1.1 Wyjaśnić, w jaki sposób różnego typu narzędzia testowe wspomagają testowanie.	K2					X									
6.2 Korzyści i ryzyka związane z automatyzacją testowania															
FL-6.2.1 Pamiętać korzyści i ryzyka związane z automatyzacją testowania.	K1					X						X			
RAZEM liczba celów nauczania pokrywających cel biznesowy		**6**	**22**	**6**	**9**	**20**	**6**	**6**	**1**	**7**	**8**	**1**	**5**	**4**	**4**

© International Software Testing Qualifications Board © Polish Quality Board

10. Dodatek C – nota wydania

Nota wydania dla sylabusa wersji 4.0.1

Sylabus ISTQB® Certyfikowany tester poziom podstawowy w. 4.0.1 to errata do sylabusa Certyfikowany tester poziom podstawowy w. 4.0. Ta errata zawiera następujące zmiany.

- Wprowadzono zmiany w sformułowaniach celów nauczania, aby zachować zgodność ze Słownikiem terminów testowych ISTQB®
- Wprowadzono zmiany tekstu w celu dostosowania go do terminów Słownika terminów testowych ISTQB®
- Uwzględniono aktualizację normy ISO 25010. Nowa wersja normy ISO 25010 została opublikowana w 2023 roku. Zmieniono w niej nazwę „użyteczność" (ang. *usability*) na „zdolność do interakcji" (ang. *interaction capability*), „przenaszalność" (ang. *portability*) na „elastyczność" (and. *flexibility*) i dodano nową charakterystykę „bezpieczeństwo" (ang. *safety*). Sylabus pozostaje przy oryginalnych nazwach charakterystyk, ale uwzględnia dodatkowo nowe nazwy użyteczności i przenaszalności w sekcji 2.2.2.
- Dodano trzy słowa kluczowe (proces testowy i śledzenie powiązań w rozdziale 1, strategia testowa w rozdziale 5).
- Wprowadzono drobne poprawki w tekście w sekcjach 1.1.2, 1.4.1, 1.4.3, 1.4.4, 1.2.2, 2.1.3, 2.1.5, 2.1.6, 2.2.2, 3.1, 3.1.2, 3.2.2, 4.2.1, 4.2.4, 5.1.1, 5.1.3, 5.1.6, 5.5, 6.2.
- Poprawiono kilka literówek i ujednolicono niektóre terminy w całym sylabusie.

Nota wydania dla sylabusa w wersji 4.0

Wersja 4.0 stanowi gruntowną aktualizację sylabusa poziomu podstawowego ISTQB® opracowaną na podstawie sylabusa Certyfikowany Tester Poziom Podstawowy w. 3.1.1 oraz sylabusa Certyfikowany Tester Tester Zwinny z roku 2014. W związku z tym nie sporządzono szczegółowego opisu wydania w podziale na rozdziały i podrozdziały, a jedynie przedstawiono podsumowanie najważniejszych zmian. Ponadto w oddzielnym dokumencie z opisem wydania przedstawiono powiązania między celami nauczania sylabusa Certyfikowany tester poziom podstawowy w wersji 3.1.1 i sylabusa Certyfikowany Tester Tester Zwinny z roku 2014 a celami nauczania w nowym sylabusie Certyfikowany tester poziom podstawowy w wersji 4.0 (ze wskazaniem celów dodanych, zaktualizowanych lub usuniętych).

Do momentu powstania niniejszego sylabusa (tj. do przełomu lat 2022 i 2023) do egzaminu na poziomie podstawowym przystąpiło ponad milion osób z ponad 100 krajów, a liczba certyfikowanych testerów na całym świecie przekroczyła 800 tys. Przy założeniu, że wszystkie te osoby przeczytały sylabus poziomu podstawowego, aby zdać egzamin, dokument ten jest prawdopodobnie najczęściej czytanym opracowaniem dotyczącym testowania oprogramowania! Obecna, gruntownie zaktualizowana wersja czerpie z tego dziedzictwa, a jednocześnie pozwala jeszcze lepiej zaprezentować jakość usług, które ISTQB® oferuje globalnej społeczności testerów–kolejnym setkom tysięcy odbiorców.

W bieżącej wersji wszystkie cele nauczania zostały przeredagowane w taki sposób, aby każdy z nich stanowił niepodzielną całość oraz aby można było jednoznacznie powiązać cele nauczania z treścią podrozdziałów sylabusa. Zadbano też o to, aby sylabus nie zawierał treści

© International Software Testing Qualifications Board © Polish Quality Board

niepowiązanych z żadnymi celami nauczania. Autorzy skupili się na możliwości praktycznego wykorzystania przedstawionych treści oraz na równowadze między wiedzą a umiejętnościami, dzięki czemu nowa wersja powinna być bardziej przystępna i zrozumiała oraz łatwiejsza do przetłumaczenia, a zawarty w niej materiał – łatwiejszy do opanowania.

W bieżącym wydaniu głównym (4.0) wprowadzono następujące zmiany:

Zmniejszono objętość całego sylabusa. Sylabus nie jest podręcznikiem, lecz dokumentem mającym na celu przedstawienie w skrócie podstawowych elementów szkolenia wprowadzającego do tematyki testowania oprogramowania, w tym wskazanie, jakie tematy należy poruszyć i na jakim poziomie. W szczególności:

- w większości przypadków z tekstu usunięto przykłady, ponieważ przygotowanie przykładów i ćwiczeń do wykorzystania w trakcie szkolenia jest zadaniem dostawcy szkoleń;
- zastosowano „listę kontrolną pisania sylabusów", która określa sugerowaną maksymalną objętość tekstu (w j. angielskim) poszczególnych celów nauczania na poszczególnych poziomach wiedzy (K1 = maks. 10 wierszy, K2 = maks. 15 wierszy, K3 = maks. 25 wierszy).

Zmniejszono liczbę celów nauczania (LO – *Learning Objectives*) w porównaniu z sylabusem Certyfikowany Tester Poziom Podstawowy w wersji 3.1.1 oraz sylabusem Certyfikowany Tester Tester Zwinny w wersji 2014:

- 14 LO na poziomie K1 w porównaniu z 21 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (15) i sylabusie dla testerów zwinnych w wersji 2014 (6);
- 42 LO na poziomie K2 w porównaniu z 53 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (40) i sylabusie dla testerów zwinnych w wersji 2014 (13);
- 8 LO na poziomie K3 w porównaniu z 15 LO w sylabusie poziomu podstawowego w wersji 3.1.1 (7) i sylabusie dla testerów zwinnych w wersji 2014 (8).

Zwiększono liczbę odwołań do klasycznych i/lub powszechnie uznanych książek i artykułów na temat testowania oprogramowania i zagadnień pokrewnych.

Istotne zmiany w rozdziale 1 (Podstawy testowania):

- Rozszerzono i poprawiono podrozdział dotyczący umiejętności w dziedzinie testowania.
- Dodano sekcję dotyczącą podejścia opartego na zaangażowaniu całego zespołu (K1).
- Przeniesiono sekcję dotyczącą niezależności testowania z rozdziału 5 do rozdziału 1.

Istotne zmiany w rozdziale 2 (Testowanie w cyklu wytwarzania oprogramowania):

- Przeredagowano i poprawiono treść sekcji 2.1.1 i 2.1.2 oraz zmodyfikowano odpowiadające im LO.
- Poświęcono więcej uwagi praktykom takim jak podejście „najpierw test" (K1), przesunięcie w lewo (K2) czy retrospektywa (K2).
- Dodano nową sekcję dotyczącą testowania w kontekście metodyki DevOps (K2).
- Podzielono testowanie integracyjne na dwa oddzielne poziomy testów: testowanie integracji modułów i testowanie integracji systemów.

© International Software Testing Qualifications Board © Polish Quality Board

Istotne zmiany w rozdziale 3 (Testowanie statyczne):

- Usunięto sekcję dotyczącą technik przeglądu wraz z LO na poziomie K3 (stosowanie technik przeglądu).

Istotne zmiany w rozdziale 4 (Analiza i projektowanie testów):

- Usunięto treść dotyczącą testowania opartego na przypadkach użycia (treść ta jest nadal dostępna w sylabusie dla analityków testów na poziomie zaawansowanym).
- Poświęcono więcej uwagi podejściu opartemu na współpracy poprzez dodanie nowego LO na poziomie K3 dotyczącego projektowania przypadków testowych metodą ATDD oraz dwóch nowych LO na poziomie K2 dotyczących historyjek użytkownika i kryteriów akceptacji.
- Materiał dotyczący testowania i pokrycia decyzji zastąpiono materiałem dotyczącym testowania i pokrycia gałęzi. Wynika to z faktu, że: po pierwsze, pokrycie gałęzi jest częściej stosowane w praktyce; po drugie, różne normy różnie definiują pojęcie „decyzji" (inaczej niż w przypadku „gałęzi"); po trzecie, pozwala to usunąć subtelną, ale poważną wadę występującą w dotychczasowym sylabusie poziomu podstawowego z 2018 r., w którym napisano, że „uzyskanie stuprocentowego pokrycia decyzji gwarantuje stuprocentowe pokrycie instrukcji kodu", co nie jest prawdą w przypadku programów, w których nie występują decyzje.
- Poprawiono podrozdział dotyczący korzyści wynikających z testowania białoskrzynkowego.

Istotne zmiany w rozdziale 5 (Zarządzanie czynnościami testowymi):

- Usunięto sekcję dotyczącą strategii testów i podejść do testowania.
- Dodano nowy LO na poziomie K3 dotyczący technik szacowania pracochłonności testowania.
- Omówiono szerzej dobrze znane pojęcia i narzędzia związane ze zwinnym wytwarzaniem oprogramowania w kontekście zarządzania testami: planowanie iteracji i wydań (K1), piramidę testów (K1) i kwadranty testowe (K2).
- Poprawiono strukturę podrozdziału dotyczącego zarządzania ryzykiem poprzez opisanie czterech głównych czynności, czyli: identyfikacji ryzyka, oceny ryzyka, łagodzenia ryzyka i monitorowania ryzyka.

Istotne zmiany w rozdziale 6 (Narzędzia testowe):

- Zmniejszono zakres treści dotyczących niektórych zagadnień związanych z automatyzacją testów, ponieważ były one zbyt zaawansowane dla poziomu podstawowego – usunięto sekcję dotyczącą wyboru narzędzi, realizacji projektów pilotażowych i wprowadzania narzędzi do organizacji.

© International Software Testing Qualifications Board © Polish Quality Board

11. Indeks

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board

© International Software Testing Qualifications Board © Polish Quality Board